National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

University of Alberta

Tree-structured Linear Cellular Automata

by

Jin Li ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science

in

Department of Electrical Engineering

Edmonton, Alberta
Fall, 1995

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file   Votre référence

Our file   Notre référence

THE AUTHOR HAS GRANTED AN IRREVOCABLE NON-EXCLUSIVE LICENCE ALLOWING THE NATIONAL LIBRARY OF CANADA TO REPRODUCE, LOAN, DISTRIBUTE OR SELL COPIES OF HIS/HER THESIS BY ANY MEANS AND IN ANY FORM OR FORMAT, MAKING THIS THESIS AVAILABLE TO INTERESTED PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE IRREVOCABLE ET NON EXCLUSIVE PERMETTANT A LA BIBLIOTHEQUE NATIONALE DU CANADA DE REPRODUIRE, PRETER, DISTRIBUER OU VENDRE DES COPIES DE SA THESE DE QUELQUE MANIERE ET SOUS QUELQUE FORME QUE CE SOIT POUR METTRE DES EXEMPLAIRES DE CETTE THESE A LA DISPOSITION DES PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP OF THE COPYRIGHT IN HIS/HER THESIS. NEITHER THE THESIS NOR SUBSTANTIAL EXTRACTS FROM IT MAY BE PRINTED OR OTHERWISE REPRODUCED WITHOUT HIS/HER PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE DU DROIT D'AUTEUR QUI PROTEGE SA THESE. NI LA THESE NI DES EXTRAITS SUBSTANTIELS DE CELLE-CI NE DOIVENT ETRE IMPRIMES OU AUTREMENT REPRODUITS SANS SON AUTORISATION.

ISBN 0-612-06567-7

Canadä

University of Alberta

Library Release Form

Name of Author: **Jin Li**

Title of Thesis: **Tree-structured Linear Cellular Automata**

Degree: **Master of Science**

Year This Degree Granted: **1995**

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided, neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

230 Woodridge Cr. #510
Ottawa, Ontario
K2B 8G2

Date: July 27, 95

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **Tree-structured Linear Cellular Automata** submitted by **Jin Li** in partial fulfillment of the requirements for the degree of Master of Science.

Sun Xiaoling

Supervisor: Dr. X. Sun

Jack Mowchenko

Examiner: Dr. J. Mowchenko

W Grover

Examiner: Dr. W. Grover

External Examiner: Dr. M. Green

Date: July 25, 1995

Abstract

This thesis formally introduces a novel implementation structure of linear finite state machines (LFSMs), named tree-structured linear cellular automata (TLCA). We define the structures and operations of TLCA and explore their applications as pseudorandom binary sequence generators in digital system testing.

We identify five types of maximum length TLCA. The algorithms to inspect maximum length TLCA are developed. The lookup tables of low cost maximum length TLCA are produced up to degree 60.

A comparative study of the pseudorandom behavior of the LFSMs, linear feedback shift registers (LFSRs), linear hybrid cellular automata (LHCA), rectangular-structured linear cellular automata (RLCA) and TLCA, is conducted. Computer simulation results reveal the relation between the pseudorandomness of the test sequences generated by the LFSMs and their test coverage for digital circuitry and are reported on the standard ISCAS '85 benchmark circuits. It suggests that maximum length TLCA be a viable alternative to both the conventional LFSRs and the more recent LHCA as pseudorandom sequence generators.

Acknowledgements

I wish to express my sincere appreciations to my supervisor, Dr. Xiaoling Sun, for her patience, understanding and encouragement. Her commendable guidance on the course of my work on this thesis is gratefully acknowledged. The financial assistance received from her research grant is also gratefully acknowledged.

I would like to thank the member of my supervisor committee, Dr. Jack Mowchenko, and my fellow graduate student, Mr. Michael Olson, for proofreading of my thesis. My thanks go to the other members of my committee, Dr. Wayne Grove and Dr. Mark Green, for their valuable comments. My thanks also go to Mr. Kevin Cattell, at the Department of Computer Science, University of Victoria, for his help in using Maple software.

I would like to thank my fellow graduate students, Yanming Li, Baolian Xu, Ketan Bhalla, and Nicole Sat, for sharing their knowledge and friendship.

Technical assistance from Norman Jantz has been an immense contribution and is greatly appreciated. Without him, the computer program work could not have been done.

Finally, a well-deserved expression of appreciation goes to my wife, Jin Wen, for her understanding, encouragement and efforts.

Contents

1 **Introduction** 1

2 **Background and Review** 6

- 2.1 Definitions 6
- 2.2 Linear Feedback Shift Registers 9
  - 2.2.1 Analysis of Shift-Register Sequences 9
  - 2.2.2 The Matrix Method 13
  - 2.2.3 Algorithm for Finding Primitive Polynomials 14
  - 2.2.4 Representations of LFSRs 16
- 2.3 One-dimensional Linear Hybrid Cellular Automata 16
  - 2.3.1 Notations and Computation Rules 17
  - 2.3.2 1-d LHCA Transition Matrix 19
  - 2.3.3 Maximum Length 1-d LCA 21
  - 2.3.4 LHCA and LFSRs with the Same Characteristic Polynomials 22
  - 2.3.5 Synthesis Algorithm 24
- 2.4 Two-dimensional Linear Cellular Automata 27
- 2.5 Fault Models 27
  - 2.5.1 Stuck-at Fault Model 29
  - 2.5.2 Stuck-open Fault Model 30
  - 2.5.3 Transition Fault Model 32

3 **Tree-Structured Linear Cellular Automata** 35

3.1 Notations and Definitions 35
  3.1.1 Computation Rules 36
  3.1.2 Regularly Structured TLCA 40
3.2 Transition Matrices of TLCA 41
3.3 Maximum Length TLCA 43
  3.3.1 Non-primitive TLCA Structures 44
  3.3.2 Primitive TLCA Structures 45
  3.3.3 Hardware Cost of TLCA Structures 54
3.4 Summary 57

4 Maximum Length TLCA **58**
4.1 Algorithm I 58
4.2 Implementation of Algorithm I 60
  4.2.1 Multiple Precision Integers 61
  4.2.2 Prime Factors of $2^n - 1$ 63
  4.2.3 Matrix Multiplication 63
  4.2.4 Powering Algorithm 64
4.3 Algorithm II 65
4.4 Minimal Cost Maximum Length TLCA 70
4.5 Maximum Length TLCA vs Primitive Polynomials 77

5 Pseudorandomness of LFSMs: Theory and Simulation **82**
5.1 Measures of Pseudorandomness 82
  5.1.1 Equidistribution Test 83
  5.1.2 Visual Test and Correlation Test 85
  5.1.3 The Role of Fault Simulation 94
5.2 Fault Simulation 95
  5.2.1 Simulation Environment 96
  5.2.2 Simulation Results 97
5.3 A Note on the Two-pattern Transition Property 114

6 Conclusion 121

Bibliography 124

Appendices 129

A Number of Primitive Polynomials and Irreducible Polynomials 129

B Implementation of Algorithm I 131

B.1 Source Code 131

B.2 Manual Page 142

C User Manual for the Fault Simulator 144

C.1 Manual Page 144

D Data of Test Length vs Fault Coverage 147

E The LFSRs, LHCA and TLCA in Fault Coverage Simulations 151

List of Figures

1.1 A signature analysis environment 2

2.1 (a) a LFSR(I) and (b) a LFSR(II) with the common characteristic polynomial $f(x) = 1 + x^3 + x^5$ 10

2.2 An example of CA 17

2.3 Naming of CA Rules 18

2.4 (a) General case; (b) irreducible machines and polynomials 23

2.5 Transition matrices of an LFSR(I) and an LHCA 25

2.6 The structure of rectangle-structured LCA of degree 4×4 28

2.7 Single stuck-at fault model 30

2.8 CMOS realization of a 2 input NAND gate 31

2.9 CMOS 2 input NAND gate with p_1 open 32

3.1 A binary tree structure 36

3.2 (a) A complete TLCA of odd degree 7 (b) A complete TLCA of even degree 6 37

3.3 A TLCA of degree 7 using rules 31, 23 and 11 38

3.4 An example of a cell in a TLCA without one input 40

3.5 Examples of non-primitive TLCA structures 46

3.6 (a) TLCA(I) structure of odd degree (b) TLCA(I) of even degree 48

3.7 TLCA(II) structure 49

3.8 TLCA(III) structure 51

3.9 (a) TLCA(IV) structure of odd degree (b) TLCA(IV) of even degree . . . 52
3.10 TLCA(V) structure . . . 54

4.1 Computational complexity ($O(n^4)$) of the irreducibility test 61

5.1 Visual test for LFSR, LHCA and TLCA of degree 20 89
5.2 Visual test for the LFSR with characteristic polynomial $x^{20} + x^3 + 1$ 90
5.3 Auto and cross correlation test for LFSR of degree 36 92
5.4 Auto and cross correlation test for LHCA of degree 36 93
5.5 Auto and cross correlation for TLCA(III) of degree 36 93
5.6 Transition fault coverage vs test length using C1355 (41 inputs) . 106
5.7 Transition fault coverage vs test length using C3540 (50 inputs) . 106
5.8 Transition fault coverage vs test length using C880 (60 inputs) . . 107
5.9 Stuck-open fault coverage vs test length using C1355 (41 inputs) . 107
5.10 Stuck-open fault coverage vs test length using C3540 (50 inputs) . 108
5.11 Stuck-open fault coverage vs test length using C880 (60 inputs) . 108

List of Tables

2.1 States of the LFSR(II) with characteristic polynomial $f(x) = 1 + x^3 + x^5$ 11

2.2 Operation of fault free NAND gate 31

2.3 Slow-to-rise fault 34

2.4 Slow-to-fall fault 34

3.1 TLCA rules 39

3.2 Computation rules of five primitive TLCA structures 47

3.3 Number of XORs in LHCA and Types I to V TLCA 56

4.1 List of minimal cost maximum-length TLCA(I) 72

4.2 List of minimal cost maximum-length TLCA(II) 73

4.3 List of minimal cost maximum-length TLCA(III) 74

4.4 List of minimal cost maximum-length TLCA(IV) 75

4.5 List of minimal cost maximum-length TLCA(V) 76

4.6 Primitive TLCA(III) and their characteristic polynomials 81

5.1 Equidistribution test for the LFSRs (Y=Yes, S=Suspect, N=No) . 86

5.2 Equidistribution test for the LHCA (Y=Yes, S=Suspect, N=No) . 87

5.3 Equidistribution test for the TLCA(III) (Y=Yes, S=Suspect, N=No) 88

5.4 Characteristics of ISCAS '85 Benchmark Circuits 97

5.5 Stuck-at fault simulation results for LFSR, LHCA and TLCA . . 100

5.6 Stuck-at fault coverages at different lengths 102

5.7 Transition fault simulation results for LFSR, LHCA and TLCA . 104

5.8 Stuck-open fault simulation results for LFSR, LHCA and TLCA . 110

5.9 Fault coverage of three types of LFSMs with different initial states 112

5.10 Characteristics of fault coverage distribution of three types of LFSMs113

5.11 Test vectors produced by the LFSR(I), LFSR(II), LHCA and TLCA(III) with characteristic polynomial $x^5 + x^3 + 1$ 116

5.12 Number of the transitions of different LFSM of degree 5 117

5.13 Number of different k-cell substate vectors with 2^{2k} transition capability 120

A.1 Number of primitive polynomials of degree n 130

D.1 Fault coverages at different number of test patterns 148

D.2 Fault coverages at different number of test patterns 149

D.3 Fault coverages at different number of test patterns 150

E.1 The LFSMs used in fault coverage simulations in Chapter 5 . . . 152

Chapter 1

Introduction

The testing of a digital system determines whether it is manufactured properly and behaves correctly. Digital testing encompasses logic and parametric tests. *Logic testing* concerns the logical correctness of a *circuit under test* (CUT), while *parametric testing* examines circuit parameters such as current, voltage, time delay and power consumption. This thesis addresses issues in logic testing. The term *testing* is used to refer to logic testing.

Testing techniques can be classified into two categories, *external test* and *built-in self-test (BIST)* [27, page 131]. External testing uses a tester external to a CUT to stimulate the circuit and evaluate the circuit responses. A general purpose tester may cost from one to several million dollars, and is not necessarily available to all designers. Moreover, a large volume of data needs to be handled by the tester, resulting in long testing times and high testing costs.

A viable alternative to traditional external testing is built-in self-test (BIST). In general, BIST refers to the inclusion of on-chip circuitry to facilitate testing so that complicated digital systems can be tested either without the need for external testers, or with reduced dependency on external hardware. As circuit density rapidly increases, testing of digital circuitry has become more and more difficult because of the increased system complexity and decreased circuit accessibility. On the other hand, very large scale integration (VLSI) and ultra large scale integration (ULSI) permit a larger portion of silicon area to be devoted to BIST circuitry, thus allowing a significant reduction on testing cost.

Testing techniques can be on-line or off-line. *On-line testing* allows the circuit to be tested as it is performing its intended functions, while *off-line testing* typically requires the CUT to suspend normal operation and enter a separate test mode. Both on-line and off-line testing can be built-in or external. Today, off-line BIST techniques are widely used in engineering practice, while on-line testing can only be found in some safety-critical systems due to the high cost in silicon and design process.

A typical testing environment consists of two key components: a mechanism to provide input stimuli to a CUT, and a mechanism to evaluate the circuit responses. The best known off-line testing technique is *signature analysis* [1, pages 432-448]. It is based on the concept of cyclic redundancy checking (CRC), and is realized in hardware using linear feedback shift registers (LFSRs) [5]. Figure 1.1 shows a signature analysis environment. The test pattern generator generates test vectors to stimulate the CUT. The circuit responses (a large volume of data) are compacted into a short *signature* (usually 16 or 32 bits) by the signature analyzer. The comparator compares the signature with the precomputed fault-free reference. An agreement (disagreement) of the two indicates that the CUT passes (fails) the test.

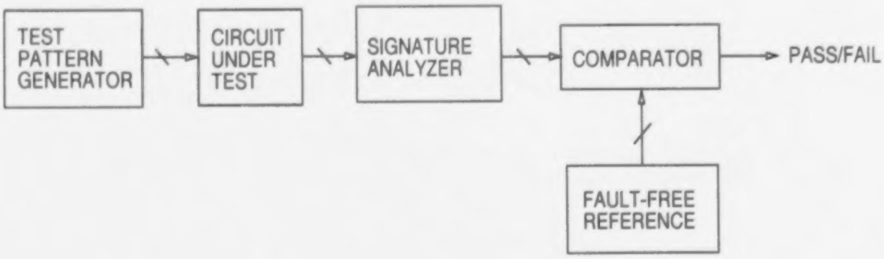


Figure 1.1: A signature analysis environment

A shift register is a collection of storage elements (for example, flip-flops) connected so that the state of each element is shifted to the next element in response to a shifting clock signal. A *linear feedback shift register* (LFSR) is a shift register with a linear feedback network using XOR gates, and is uniquely defined by a polynomial in binary field [1]. LFSRs are commonly used as test pattern generators and data compactors [27, pp. 131-153].

Exhaustive test pattern generation generates all the 2^n possible input combinations for a circuit of n inputs. However, it is only applicable to circuits with a small number of inputs (i.e. when n is in the lower twenties). In practice, commercial circuits with over 100 inputs are not uncommon, and the affordable testing time for circuits is in seconds. Generating exhaustive test patterns for a circuit with 100 inputs would require several days, even if the fastest computer available today is used.

Digital testing has two basic requirements, the lowest possible test cost and the highest possible test coverage. Test cost is a function of test time, which is largely dependent on the test length of chosen test vectors. The test coverage, on the other hand, determines the quality of digital circuits and requires sufficiently long test sets. Therefore, tradeoff between the two requirements has to be made. It is desirable to choose a small subset of all the possible input combinations, which requires a reasonable test time and is able to detect the maximum number of faults in a CUT. There are two alternative approaches, *random* and *pseudorandom* test pattern generation. A random test sequence has a high probability of exercising all input lines with different logic values. However, it suffers two main problems: it is hard to determine the test coverage, and the test sequence is not repeatable [1]. In manufacture testing, a million copies of a circuit may be fabricated. Therefore, a repeatable test sequence would provide a standard testing measurement on the circuit and simplify the testing procedures.

Pseudorandom test pattern generation generates a subset of 2^n possible test vectors in a deterministic fashion. The vectors have many characteristics of random patterns, i.e. the vectors appear to be random in the local sense, but they are repeatable. Various hardware implementations of pseudorandom test pattern generators have been proposed. LFSRs are the most commonly used implementation structure. A brief review of the implementation structures and the theoretical background can be found in Section 2 of this thesis. A comprehensive exposition of this subject can be found in [5].

It should be pointed out that the application of pseudorandom sequences is not limited to digital testing. In fact, they have found wide use in communication systems, encipherment, error-correcting coding and cryptography.

The primary goal of this thesis is to explore alternative implementation structures for pseudorandom sequence generation so that test coverage of digital circuitry can be improved and test time can be reduced. There are two basic related issues: (1) to find a suitable structure and implementation of a new binary sequence generator, and (2) to study its pseudorandom behavior and determine its test coverage in testing applications.

In the past ten years, considerable interest has been developed in the behavior of *cellular automata* (CA). The initial work was pioneered by Wolfram [37] and later by Pries *et al.* [28]. It has been found that *one-dimensional linear hybrid cellular automata* (LHCA) are of special interest as an alternative source of test stimuli to the conventional LFSR implementations [10, 30]. It is reported that LHCA with maximum length cycle have superior performance as pseudorandom test pattern generators to LFSRs [39]. Industrial companies have reported the use of LHCA in their built-in test circuitry for commercial products [26].

Janowalla defined and explored the behavior of *two-dimensional* machines, called rectangle-structured linear cellular automata (RLCA) [23]. The initial result showed that the one-dimensional LHCA and the two-dimensional RLCA perform equally well as pseudorandom test pattern generators, and that both are much better than LFSRs. However, RLCA cannot be implemented at every degree. For instance, RLCA of prime number degrees cannot be implemented [23].

The objective of this thesis is to formally introduce *two-dimensional tree-structured linear cellular automata* (TLCA), and explore their applications in pseudorandom pattern generation. The research will increase our understanding of the pseudorandom behavior of linear finite state machines, and the relation between the pseudorandom measurements and test coverage. It will also prepare the background for theoretical treatment of measuring testing quality in future studies.

The remainder of this thesis is organized as follows. Chapter 2 prepares the general background relevant to this thesis. Additional details are introduced as necessary in later chapters. We review the previous work in LFSR, LHCA and RLCA, and examine their merits and demerits in testing applications.

Chapter 3 introduces the definitions, structures and computation rules of TLCA. The transition matrix of TLCA and five maximum length TLCA structures are formally defined.

Chapter 4 presents the computer algorithms to search maximum length TLCA, and discusses the problems and the solutions in their C programming implementation. Lookup tables of minimal cost maximum length TLCA of degree 2 to 60 are given.

Chapter 5 explores the application of TLCA as pseudorandom test pattern generators, and examines their effects on test coverage under various fault models. Fault simulations are conducted on the standard ISCAS '85 benchmark circuits, and the statistical results of the simulations are reported. Concluding remarks and topics of future research are presented in Chapter 6.

There are five appendices. Appendix A lists the number of all primitive polynomials of each degree, up to degree 24. Appendix B contains the source code for the implementation of Algorithm I described in Section 4.1. Appendix C is the user manual of the fault simulator used in Chapter 5. Appendix D provides the tabular data of the fault coverage as functions of test length. The plots of the data can be found in Figures 5.6, 5.7, 5.8, 5.9, 5.10, and 5.11 of Chapter 5. Appendix E shows the actual LFSMs used as the test pattern generators in the fault simulations reported in Chapter 5.

Chapter 2

Background and Review

Problems in pseudorandom sequence generation and its applications in digital testing span several subjects of science and engineering. In the first four sections of this chapter, we prepare the necessary mathematical and practical background required for the study of tree-structured linear cellular automata (TLCA) in Chapters 3 and 4. We review the extensive work on structures and operations of pseudorandom sequence generators. We summarize their representations used in different contexts, and provide the insight of the correlations among them.

To prepare for the study of the effectiveness of pseudorandom sequences on test coverage for digital systems in Chapter 5, fault models that represent faulty behavior of digital components are introduced in Section 2.5.

2.1 Definitions

Definition 2.1 A nonzero integer t is a *divisor* of an integer s if there is an integer u such that $s = tu$. In this case, we write $t|s$.

Definition 2.2 A *prime* is a positive integer greater than 1 whose only positive divisors are 1 and itself.

Euclid's Lemma: If p is a prime then $p|ab$ implies $p|a$ or $p|b$.

Definition 2.3 *Modular Equations:* If a and b are integers and n is a positive integer, we write $a = b \bmod n$ when n divides $a - b$. E.g., $13 = 3 \bmod 5, 22 = 10 \bmod 6, -10 = 4 \bmod 7$.

Definition 2.4 [7, pp.27] A *field* F is a set that has two operations: addition and multiplication, defined on it such that the following axioms are satisfied:

1. The set is an abelian group under addition.
2. The field is closed under multiplication, and the set of nonzero elements is an abelian group under multiplication.
3. The distributive law

$$(a + b)c = ac + bc$$

holds for all a, b, c in the field.

A field with finite elements is called a *finite field*, or a *Galois field*. It is denoted by the label $GF(q)$ with q elements in a field. $GF(2)$ has elements 0 and an 1, which are under modulo-2 addition and modulo-2 multiplication.

Definition 2.5 Given a sequence of numbers, $a_0, a_1, a_2, \cdots, a_n, \cdots$, a *generating function* $G(x)$ can be associated with the sequence by the rule

$$G(x) = a_0 + a_1 x + a_2 x_2 + \cdots + a_n x_n + \cdots . \tag{2.1}$$

Definition 2.6 [33, p. 297] A function f: $V \rightarrow V'$ is a *linear function* from a vector space V into a vector space V' over the same scalar field q if, for all c_1 and c_2 in q and v_1 and v_2 in V,

$$f(c_1 \cdot v_1 \; + \; c_2 \cdot v_2) \; = \; c_1 \cdot f(v_1) \; + \; c_2 \cdot f(v_2) \, ,$$

where "·" denotes multiplication of a scalar with a vector, and "+" is the operation defined in V. Here, we consider the field $GF(2)$.

Definition 2.7 A machine M is a *linear finite state machine* (LFSM) [33, pp. 297-298] if:

1. the state space S_M of M, the input space I_M, and the output space Y_M are each vector spaces over a finite field K;
2. let the vector s_i^t denote the state of the machine at time t, the vector u_i denote the input to the machine, and the vector y_i denote the output of the machine. The next state s_i^{t+1} of M at time $t+1$ is defined by

   $$s_i^{t+1} = A \cdot s_i^t + B \cdot u_i \tag{2.2}$$

   and the output is defined by

   $$y_i = C \cdot s_i^t + D \cdot u_i \ , \tag{2.3}$$

   where A, B, C, and D are transformation matrices over the finite field K, and "$\cdot$" denotes matrix multiplication.

The machine is *linear* because the transition function from the current state to the next state is defined to be linear, and thus can be represented by a matrix with the elements in the appropriate field (here $GF(2)$).

Definition 2.8 *Autonomous linear finite state machines* are linear finite state machines without external input u_i.

For an autonomous LFSM there is no second term in equation (2.2) and (2.3) and therefore the next state function is reduced to

$$s_i^{t+1} = A \cdot s_i^t \ . \tag{2.4}$$

The matrix A is called the *state transition matrix.* Thus with the definition of linearity, the state of the machine at time $t+1$ linearly depends on the state of the machine at time t, and the transformation between states can be accomplished by matrix multiplication with A. In this thesis we only consider autonomous LFSM.

2.2 Linear Feedback Shift Registers

2.2.1 Analysis of Shift-Register Sequences

Let the sequence $\{a_m\} = \{a_0,\ a_1,\ a_2,\ \cdots\}$ represents the history of the output stage of a shift register where $a_i, \in \{0,1\}$, depends on the state of the output stage at time t_i. The properties of this shift-register sequence can be examined by analysis of the generating function created by the rule

$$G(x) = \sum_{m=0}^{\infty} a_m x^m \ . \tag{2.5}$$

Because the contents of an n-stage shift register eventually get shifted out to the right to become the sequence under analysis, the initial state of the shift register may be thought of as

$$a_{-n}, a_{-n+1}, \cdots, a_{-2}, a_{-1} \ .$$

If the recurrence relation defining $\{a_m\}$ is

$$a_m = \sum_{i=1}^{n} c_i a_{m-i} \ , \tag{2.6}$$

where $c_i \in \{0,1\}$ depending on the feedback, then $G(x)$ becomes [5]

$$G(x) = \frac{c_n}{1 - \sum_{i=1}^{n} c_i x^i} \ . \tag{2.7}$$

Definition 2.9 The *characteristic polynomial* of a sequence $\{a_m\}$ produced by an LFSR is defined as

$$f(x) = 1 - \sum_{i=1}^{n} c_i x^i = \sum_{i=0}^{n} c_i x^i \ . \tag{2.8}$$

Note that $c_n = 1$ for an n-stage linear feedback shift register (LFSR) with a characteristic polynomial of degree n. The relation between a generating function and a characteristic polynomial is

$$G(x) = \frac{1}{f(x)} \ . \tag{2.9}$$

Figure 2.1 shows two types of LFSR implementations, named Type I and Type II LFSRs, based on the same characteristic polynomial $x^5 + x^3 + 1$. The difference between them is the placement of the XOR gates. For convenience, the Type I and II LFSRs are labeled LFSR(I) and LFSR(II), respectively.

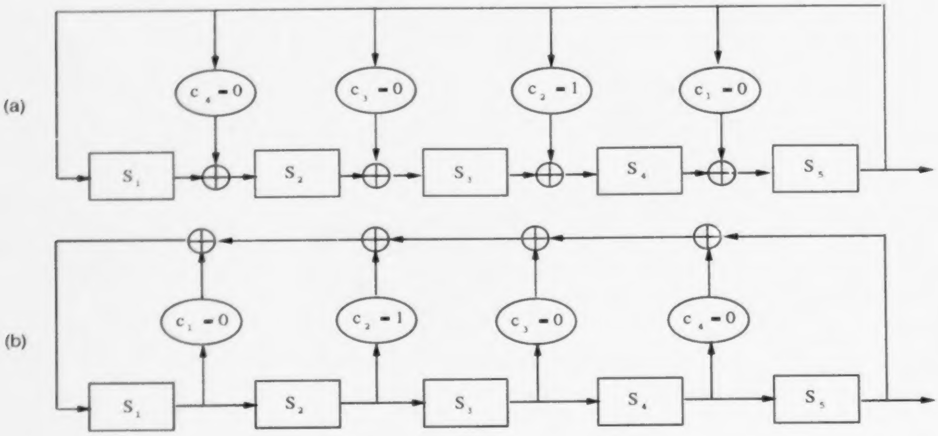


Figure 2.1: (a) a LFSR(I) and (b) a LFSR(II) with the common characteristic polynomial $f(x) = 1 + x^3 + x^5$

Example 2.1 Consider a shift-register sequence with a recursion relation

$$a_m = a_{m-3} + a_{m-5} , \tag{2.10}$$

the characteristic polynomial of this recursion relation is

$$f(x) = 1 + x^3 + x^5 , \tag{2.11}$$

which is shown in Figure 2.1. □

Table 2.1 lists the sequence of 31 states that the LFSR(II) of Figure 2.1 (b) cycles through, starting with the initial state 00001. The columns 2 to 6 list the states in binary (where s_5 is the most significant bit), and the column 1 contains the corresponding decimal representations.

State	s_5	s_4	s_3	s_2	s_1
1	0	0	0	0	1
2	0	0	0	1	0
4	0	0	1	0	0
9	0	1	0	0	1
18	1	0	0	1	0
5	0	0	1	0	1
11	0	1	0	1	1
22	1	0	1	1	0
12	0	1	1	0	0
25	1	1	0	0	1
19	1	0	0	1	1
7	0	0	1	1	1
15	0	1	1	1	1
31	1	1	1	1	1
30	1	1	1	1	0
28	1	1	1	0	0
24	1	1	0	0	0
17	1	0	0	0	1
3	0	0	0	1	1
6	0	0	1	1	0
13	0	1	1	0	1
27	1	1	0	1	1
23	1	0	1	1	1
14	0	1	1	1	0
29	1	1	1	0	1
26	1	1	0	1	0
21	1	0	1	0	1
10	0	1	0	1	0
20	1	0	1	0	0
8	0	1	0	0	0
16	1	0	0	0	0

Table 2.1: States of the LFSR(II) with characteristic polynomial $f(x) = 1+x^3+x^5$

One *period* of the sequence $\{a_m\}$ $(m = 31)$ that the fifth stage of the LFSR(II) in Figure 2.1 (b) generates can be seen to be

$$\begin{aligned} \{a_m\} &= a_0, a_1, a_2, \cdots, a_{m-1} \\ &= 0000100101100111110001101110101 \ . \end{aligned} \tag{2.12}$$

Definition 2.10 [5, pp. 70] If the sequence generated by an n-stage LFSR has *period* of $2^n - 1$, it is called a ***maximum-length sequence.*** The characteristic polynomial of a maximum-length sequence is called a *primitive polynomial.*

Definition 2.11 Define the *reciprocal polynomial* $f^*(x)$ of $f(x)$ as

$$f^*(x) = x^n f\left(\frac{1}{x}\right) = \sum_{i=0}^{n} c_i x^{n-i} \ . \tag{2.13}$$

Note that if a characteristic polynomial $f(x)$ defines a maximum-length sequence (i.e. is a primitive polynomial), its reciprocal will also yield a maximum-length sequence. The reciprocal polynomial is simply the time reverse of the sequence corresponding to $f(x)$ [5].

Theorem 2.1 [19, pp. 32]

Given an n-stage LFSR with the initial conditions

$$a_{-1} = a_{-2} = \cdots = a_{1-n} = 0; a_{-n} = 1, \tag{2.14}$$

then the LFSR sequence $\{a_m\}$ is periodic with a period which is the smallest integer k for which $f(x)$ divides $1 - x^k$.

Definition 2.12 If f is a polynomial which has no divisors except scalars and scalar multiples of itself, then f is said to be an *irreducible polynomial.*

Theorem 2.2 [19, pp. 33] If the sequence $\{a_m\}$ has maximum length, its characteristic polynomial is irreducible.

Theorem 2.3 [19, pp. 37] If a sequence $\{a_m\}$ is derived from an n-stage LFSR with irreducible characteristic polynomial $f(x)$, then the period of the sequence $\{a_m\}$ is a factor of $2^n - 1$.

A primitive polynomial is a special case of an irreducible polynomial. Clearly, if $2^n - 1$ is prime, every irreducible polynomial of degree n corresponds to a maximum length LFSR. The irreducibility is a necessary condition for primitive polynomials. The relationship between primitive polynomials and irreducible polynomials is

$$\text{Primitive polynomial} \underset{?}{\rightleftarrows} \text{Irreducible polynomial} \quad .$$

The primitive *trinomials*, polynomials with three nonzero coefficients (corresponding to minimal cost LFSRs), of degree up to 500 have been found and published in [4, 5, 8, 21, 32, 36].

2.2.2 The Matrix Method

Each state of an n-stage shift register can be considered as an n-dimensional vector. The shift register is then a linear operator which changes each state into the next. It is a familiar fact that a linear operator, operating on n-dimensional vectors, is most conveniently represented by an $n \times n$ matrix,

$$\left(s_0^+, s_1^+, s_2^+, \cdots, s_{n-2}^+, s_{n-1}^+\right)^T = M \cdot \left(s_0, s_1, s_2, \cdots, s_{n-2}, s_{n-1}\right)^T , \qquad (2.15)$$

where s_i denotes the present state, s_i^+ the next state, and the superscript T means the transpose of the matrices.

In general, a state transition matrix M of an n-stage LFSR takes the form

$$M = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 & 1 \\ 1 & 0 & \cdots & 0 & 0 & c_1 \\ 0 & 1 & \cdots & 0 & 0 & c_2 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 & c_{n-2} \\ 0 & 0 & \cdots & 0 & 1 & c_{n-1} \end{pmatrix} \qquad (2.16)$$

with 1's along the diagonal below the main diagonal, and the "feedback coefficients" down the last column for the LFSR(I), and

$$M = \begin{pmatrix} c_1 & c_2 & \cdots & c_{n-2} & c_{n-1} & 1 \\ 1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 0 \end{pmatrix} \tag{2.17}$$

with 1's along the diagonal below the main diagonal, and the "feedback coefficients" in the first row for the LFSR(II).

The characteristic equation of matrix M for LFSR(II) is

$$\begin{aligned} \Delta &= det(\lambda I + M) \\ &= \begin{vmatrix} \lambda + c_1 & c_2 & \cdots & c_{n-2} & c_{n-1} & 1 \\ 1 & \lambda & \cdots & 0 & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & \lambda & 0 \\ 0 & 0 & \cdots & 0 & 1 & \lambda \end{vmatrix} \\ &= \lambda^n + c_1\lambda^{n-1} + c_2\lambda^{n-2} + \cdots + c_{n-1}\lambda + 1 \;, \end{aligned} \tag{2.18}$$

which is the characteristic polynomial of the shift register.

To determine the periodicity of an LFSR is equivalent to finding the smallest power p of its transition matrix M such that $M^p = I$ [19] (where I is the identity matrix), if the matrix is non-singular[1].

2.2.3 Algorithm for Finding Primitive Polynomials

One of the most practical methods of finding primitive polynomials is the sieve method [5, 19, 32]. A series of tests is applied to determine the irreducibility of a trial polynomial of degree n, then the primitivity.

---

[1] Note: a square matrix A is said to be non-singular if and only if its determinant $|A| \neq 0$.

Test 1 A polynomial with all even exponents is a square and hence reducible. Therefor, the first test rejects all polynomials with all even exponents, such as, $x^6 + x^2 + 1 = (x^3 + x + 1)^2$.

Test 2 An irreducible polynomial has an odd number of terms, one of which is a constant. Thus the second test rejects all polynomials that fails this simple criterion.

Test 3 The maximum-length shift-register sequence has a period of $2^n - 1$. It is shown [25] that if the trial polynomial $g(x)$ is irreducible (a necessary condition for it to be primitive), it divides $1 - x^{2^n-1}$. In other words, letting $f = 2^n - 1$, if

$$x^f \neq 1 \bmod g(x)$$

then $g(x)$ is reducible and is discarded in favor of another trial polynomial. If, however, $x^f = 1 \bmod g(x)$, the tests continue since the trial polynomial is irreducible and possibly primitive.

Test 4 It must now be determined if there is a prime factor p of f such that $x^{f/p} = 1 \bmod g(x)$. If no such prime factor can be found, $g(x)$ is primitive.

Example 2.2 The sieve method is applied to each item of a list of several trial polynomials to determine their primitivity:

$$\begin{aligned} p_5(x) &= x^4 + x + 1 \\ p_6(x) &= x^4 + x^3 + 1. \end{aligned}$$

Polynomials $p_5(x)$ and $p_6(x)$ are reciprocal, so what holds for $p_5(x)$ will also hold for $p_6(x)$. Test 1 and Test 2 are clearly satisfied by $p_5(x)$. Test 3 is satisfied: $x^{15} = 1 \bmod p_5(x)$, so it is irreducible. Continuing to Test 4, only x^5 must be reduced mod $p_5(x)$. The result is $x^5 = (x^2 + x) \bmod p_5(x)$. Therefore, $p_5(x)$ and $p_6(x)$ are primitive. □

2.2.4 Representations of LFSRs

There are three different representations of LFSRs: polynomials over $GF(2)$, binary string representation, and the LFSR implementation of polynomials. Each representation provides a convenient expression in a corresponding domain, and can be easily transformed to either of the other two. A polynomial can be directly mapped into an LFSR(I) implementation, where the non-zero coefficients correspond to the feedback taps of the LFSR; and it can also be mapped to a binary string, where the non-zero and zero coefficients correspond to 1's and 0's respectively, that is, $[1c_1c_2 \cdots c_{n-1}1]$ in the equation 2.18. For example, the polynomial $x^5 + x^3 + 1$ in Figure 2.1 can be represented by [101001].

2.3 One-dimensional Linear Hybrid Cellular Automata

Cellular Automata (CA)[2] are defined as uniform arrays of identical cells in an n-dimensional space [28]. Each cell is capable of existing in a finite discrete state space, in our case a binary state space. Further, each cell is restricted to local neighborhood interaction only and has no global communication, i.e. the neighborhood of a cell is typically taken to be the cell itself and all immediately adjacent cells. The algorithm the cell uses to compute its successor state, based on the information received from its nearest neighbors, is referred to as the *computation rule* [37]. Cellular automata can be characterized by the four following properties:

1. the cellular geometry;
2. the neighborhood specification, where cells are restricted to local neighborhood interaction and have no global communication;

---

[2] CA is used as an abbreviation for both Cellular Automaton and Cellular Automata, depending on the context. Cellular Automaton indicates a single machine made up of a collection of cells whereas Cellular Automata contain more than one Cellular Automaton.

3. the number of states per cell;

4. the algorithm to compute the successor state, called its *computation rule*, based on the information received from its nearest neighbors.

2.3.1 Notations and Computation Rules

CA are finite state machines. The next state of a k-neighbor cell depends on the present states of itself and its $k-1$ neighbors, specified by its neighborhood function.

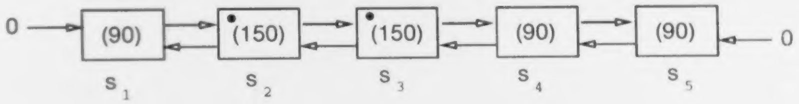


Figure 2.2: An example of CA

Figure 2.2 is an example of the type of machine being described. Let s_i^t be the state of the cell s_i at time t and the next state s_i^{t+1} of the cell s_i at time $t+1$. For a 3-neighbor cell, s_i^{t+1} can be represented as a function of the present states of cells s_i, s_{i-1} (left neighbor) and s_{i+1} (right neighbor) at time t:

$$s_i^{t+1} = f(s_{i-1}^t,\ s_i^t,\ s_{i+1}^t) \quad 1 \leq i \leq n$$

where f is known as the rule of the cell denoting the logic, and n is the number of cells or the *length* or *degree* of the CA. There are two possible boundary conditions, null or cyclic. Under a null boundary condition, a constant 0 is applied from outside, and a cyclic boundary condition requires a connection of the two ends.

If the function $f(s_{i-1}^t, s_i^t, s_{i+1}^t)$ can be expressed in the form

$$f(s_{i-1}^t, s_i^t, s_{i+1}^t) = k_1 s_{i-1}^t \oplus k_2 s_i^t \oplus k_3 s_{i+1}^t$$

where each of the constants k_i is either 0 or 1, and where $\oplus$ denotes addition over $GF(2)$, the machine is called a *linear* CA (LCA). Since a cell in a one-dimensional LCA has at most 3 neighbors, there are $2^3 = 8$ possible linear rules.

For a 3-neighbor cell containing binary states, there can be a total of $2^{2^3} = 256$ distinct binary rules (linear and non-linear rules). The rules by which a cell determines its next state were named by Wolfram [37] as follows. The states of the cell and its neighbors are written as 8 triplets in ascending (right to left) binary order as shown in Figure 2.3. The leftmost bit of a triplet shows the current state of the left neighboring cell, the middle bit shows the current state of the the cell under consideration, and the rightmost bit shows the current state of the right neighbor. The next state of the cell under consideration is written below each triplet. The resulting group of 8 bits forms a binary number between 0 and 255; the least significant bit is the one formed from the triplet 000 and the most significant bit is the one formed from 111. This binary number is used as the rule, such as, for rule 90, 01011010_2 is decimal number 90_{10}, and for rule 150, $10010111_2 = 150_{10}$. Figure 2.3 shows the rules for two of the possible connections. In rule 90, the next state is the modulo-2 sum of the present states of the two neighbors . Rule 150 derives the next state from the modulo-2 sum of the present states of the cell and both its left and right neighbors.

	7	6	5	4	3	2	1	0
Triplets	111	110	101	100	011	010	001	000
Rule 90	0	1	0	1	1	0	1	0
Rule 150	1	0	0	1	0	1	1	1
	128	64	32	16	8	4	2	1

Figure 2.3: Naming of CA Rules

The computation rules 90 and 150 are defined as follows:

$$\begin{aligned} Rule\ 90: &\ s_i^+ = s_{i-1} \oplus s_{i+1} \\ Rule\ 150: &\ s_i^+ = s_{i-1} \oplus s_i \oplus s_{i+1} \end{aligned} \tag{2.19}$$

If the same rule is applied to all the cells of an LCA, then the LCA is called a *homogeneous* LCA; if different rules are applied to the cells, then the LCA is called a *hybrid* LCA (LHCA). Figure 2.2 shows a 5-cell one-dimensional null boundary LHCA using rules 90 and 150 with the characteristic polynomial $x^5 + x^3 + 1$.

2.3.2 1-d LHCA Transition Matrix

The states of an n-cell LHCA (or an LHCA of degree n) at time t are represented by the vector $S_t = [s_1^t, s_2^t, \cdots, s_n^t]$, which also represents the pattern generated by the LHCA at time t. With the representation of the state of the LHCA, we need a linear operator to describe the computational rules that specify the state transitions of the finite state machine. The linear operator can be represented by an $n \times n$ square matrix. The next state of the LHCA can be obtained by multiplying the state vector representation of a present state by this matrix, where all operations are modulo-2. The matrix is referred to as the transition matrix T of the LHCA.

The state of a LHCA at time $t + 1$ is represented as

$$[S_{t+1}] = [T][S_t] \ ,$$

and at time $t + k$ as

$$[S_{t+k}] = [T]^k[S_t] \ .$$

Example 2.3 In Figure 2.2, the next state equations are

$$\begin{aligned} s_1^+ &= s_2 \\ s_2^+ &= s_1 \oplus s_2 \oplus s_3 \\ s_3^+ &= s_2 \oplus s_3 \oplus s_4 \\ s_4^+ &= s_3 \oplus s_5 \\ s_5^+ &= s_4 \ , \end{aligned} \tag{2.20}$$

and the corresponding state transition matrix is

$$T = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \qquad (2.21)$$

□

In general, the i-th row of a transition matrix T designates the computational rule performed by the cell s_i (numbered from left to right). For a k-neighbor cell of a LHCA there are at most k nonzero entries in the corresponding row of the matrix. A transition matrix is constructed by

1. Elements on the main diagonal, $T_{i,i}$, are denoted as d_i for $1 \leq i \leq n$ and $d_i \in (0, 1)$, where $d_i = 1$ indicates the dependency of the next state of the cell s_i on its present state (e.g., rule 150), and $d_i = 0$ means no such dependency (such as, rule 90);
2. Elements on the super-diagonal, $T_{i,i+1}$, are denoted as r_i for $1 \leq i \leq n-1$ and $r_i \in (0, 1)$. Similarly, $r_i = 1$ ($r_i = 0$) indicates that the next state of the cell s_i depends (does not depend) on the present state of its immediate right neighbor, the cell s_{i+1};
3. Elements on the sub-diagonal, $T_{i,i-1}$, are denoted as l_i for $2 \leq i \leq n$ and $l_i \in (0, 1)$. $l_i = 1$ and $l_i = 0$ represent the dependency and independency of the next state of the cell s_i on the present state of its immediate left neighbor, cell s_{i-1};
4. The other elements of T are equal to 0.

Example 2.4 The transition matrix T of 4-cell null boundary LHCA is

$$T = \begin{bmatrix} d_1 & r_1 & 0 & 0 \\ l_2 & d_2 & r_2 & 0 \\ 0 & l_3 & d_3 & r_3 \\ 0 & 0 & l_4 & d_4 \end{bmatrix}.$$

□

If the rules 90 and 150 are used, then $l_i = r_i = 1$, for all i, all the elements on the super and sub diagonals are all equal to 1. For convenience, a binary string $[d_1 d_2 \cdots d_n]$ is used to represent an LHCA of degree n. For example, the 5-cell, null boundary LHCA shown in Figure 2.2, consisting of the following rules from the left to the right: [90 150 150 90 90], can be represented by [01100]. A dot in the cell indicates that the next state of the cell depends on its own present state, that is, $d_i = 1$.

2.3.3 Maximum Length 1-d LCA

An n-cell LCA is said to have a *maximum length* if all $2^n - 1$ nonzero distinct n-stage patterns it generates appear as successive states in a single connected cycle, or if the output sequence it generates has period of $2^n - 1$. An n-cell maximum length LCA is characterized by the following relation on its transition matrix:

$$T^m = I \quad \text{for which the smallest value of } m \text{ is } 2^n - 1.$$

The characteristic polynomial of the transition matrix T of a maximum length LCA is a primitive polynomial.

There are 8 possible distinct rules for one-dimensional linear cellular automata. However, only null boundary LHCA with hybrid rules 90 and 150 cells have maximum length cycle [10, 30]. It has been shown [3, 10] that LHCA with cyclic boundary conditions at both ends have no maximum length sequence. For LHCA with the boundary condition, null at one end and cyclic at another, it has been found that only some of them have maximum length sequences [3]. In this thesis only LHCA with null boundary condition are considered.

2.3.4 LHCA and LFSRs with the Same Characteristic Polynomials

As stated in section 2.2.2, there is a 1-to-1 correspondence between a LFSR and a polynomial. In other words, given a LFSR, a unique characteristic polynomial can be determined and *vice versa*.

In general, the transition matrix T of a degree n LHCA is in the form

$$T_n = \begin{pmatrix} d_1 & 1 & 0 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & d_2 & 1 & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 1 & d_3 & 1 & 0 & \cdots & \cdots & \cdots & \cdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \cdots & \cdots & \cdots \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & 1 & d_n \end{pmatrix}. \qquad (2.22)$$

The characteristic polynomial is defined as $\Delta_n(x) = det(T_n + xI)$. For a given n-cell LHCA with $[d_1 d_2 \cdots d_n]$, Serra *et al* [30] demonstrate that the characteristic polynomial $\Delta_n(x)$ for the LHCA can be calculated by the recurrence relation

$$\Delta_k(x) = (x + d_k)\Delta_{k-1}(x) + \Delta_{k-2}(x), \qquad k \geq 1 \qquad (2.23)$$

with initial conditions $\Delta_0(x) = 1$ and $\Delta_{-1}(x) = 0$.

Example 2.5 For the LHCA [01100] in Figure 2.2, the characteristic polynomial is computed by equation (2.23) as follows.

$$\begin{aligned} \Delta_{-1}(x) &= 0 \\ \Delta_0(x) &= 1 \\ \Delta_1(x) &= (x + d_1)\Delta_0(x) + \Delta_{-1}(x) \\ &= (x + 0)1 + 0 \end{aligned}$$

$$
\begin{aligned}
&= x \\
\Delta_2(x) &= (x+d_2)\Delta_1(x)+\Delta_0(x) \\
&= (x+1)x+1 \\
&= x^2+x+1 \\
\Delta_3(x) &= (x+d_3)\Delta_2(x)+\Delta_1(x) \\
&= (x+1)(x^2+x+1)+x \\
&= x^3+x+1 \\
\Delta_4(x) &= (x+d_4)\Delta_3(x)+\Delta_2(x) \\
&= (x+0)(x^3+x+1)+x^2+x+1 \\
&= x^4+1 \\
\Delta_5(x) &= (x+d_5)\Delta_4(x)+\Delta_3(x) \\
&= (x+0)(x^4+1)+x^3+x+1 \\
&= x^5+x^3+1
\end{aligned}
$$

Therefore, the characteristic polynomial of the LHCA is x^5+x^3+1. □

In comparison, the relations between LHCA and polynomials are more complicated. As shown in [10], more than one correspondence may exist between a polynomial and an LHCA. When irreducibility is concerned, it has been found that any irreducible polynomial can be mapped into two irreducible LHCA. Figure 2.4 depicts the pictorial representation of these relations.

(a) LHCA $\xleftrightarrow{\text{not 1:1}}$ Polynomials $\xleftrightarrow{\text{1:1}}$ LFSRs

(b) irreducible LHCA $\xleftrightarrow{\text{2:1}}$ irreducible polynomials $\xleftrightarrow{\text{1:1}}$ irreducible LFSRs

Figure 2.4: (a) General case; (b) irreducible machines and polynomials

Theorem 2.4 [30, pp. 771] If two matrices A and B have the same characteristic polynomial $\Delta_n(x)$ which is irreducible (or primitive), then A is similar to B. In fact, there exists a non-singular matrix P such that $B = P^{-1}AP$, whose operation is called a similarity transform.

Theorem 2.5 [30, pp. 771] An LCA and an LFSR with the same irreducible (or primitive) characteristic polynomial are isomorphic.

It is known that isomorphic LCA and LFSRs have identical cycle structures. The proof is based on the properties of group theory [33] where an element h is the conjugate of g in a group G if there exists an x in G such that $h = x \cdot g \cdot x^{-1}$. Conjugate elements h and g have the same cycle structure in all permutation representations of the group.

Example 2.6 Figure 2.5 shows an LHCA and an LFSR with their corresponding transition matrix and their characteristic polynomial. The cycle structure is also shown in the state transition diagrams. Since this polynomial is irreducible, but not primitive, the states form four separate cycles, where state 0 always goes back to itself. □

Theorem 2.5 has another meaning, i.e. the characteristic polynomial of an LHCA determines the recurrence relation, similar to equation (2.10), that is displayed in the output sequence of a single output stage. By the similarity, the output sequence of the LHCA is identical to that of a stage of the similar LFSR. It is easy to verify that the recurrence relation for the sequence $\{a_m\}$ generated by a cell of the LHCA in Figure 2.2 is also $a_m = a_{m-3} + a_{m-5}$, comparing with that of the LFSR in Figure 2.1.

2.3.5 Synthesis Algorithm

The synthesis of an LHCA for a polynomial $p(x)$ is the process of constructing the LHCA that has the $p(x)$ as its characteristic polynomial over $GF(2)$. Cattell and Muzio [12] have developed an LHCA synthesis algorithm based on Euclidean Division.

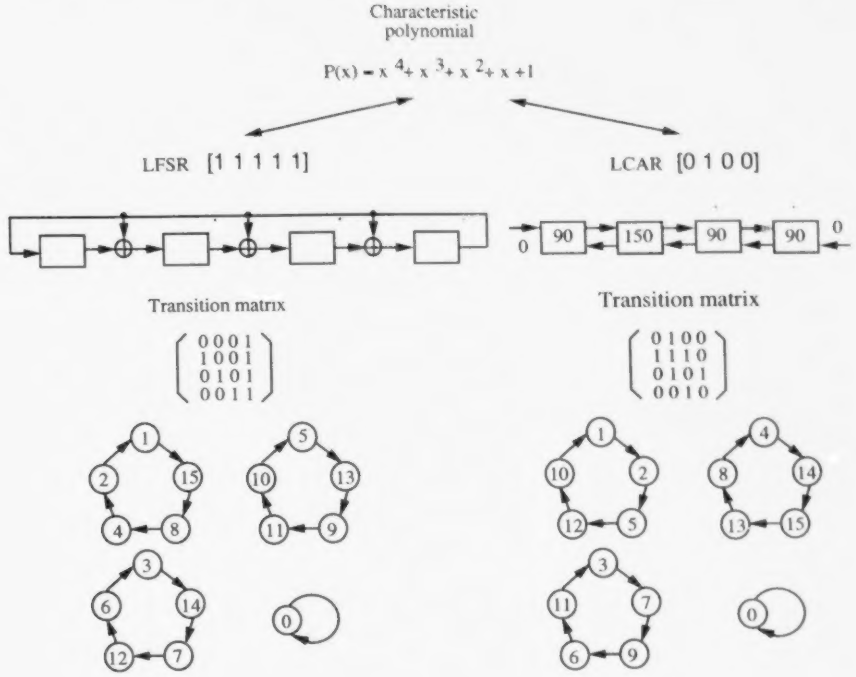


Figure 2.5: Transition matrices of an LFSR(I) and an LHCA

The Euclidean division algorithm for polynomials states that for given polynomials $a(x)$ and $b(x)$ with $b(x) \neq 0$, there exist unique polynomials $q(x)$ and $r(x)$ such that

$$a(x) = q(x)b(x) + r(x) \; ,$$

where either the degree of $r(x)$ is less than the degree of $b(x)$, or $r(x) = 0$. The polynomial $a(x)$ is the *dividend*, $b(x)$ is the *divisor*, $q(x)$ is the *quotient*, and $r(x)$ is the *remainder*.

In general, the division algorithm to compute a $[d_1 d_2 \cdots d_n]$ of an LHCA with a given irreducible polynomial $\Delta_n(x)$ is summarized as

$$\begin{aligned} \Delta_n(x) &= (x + d_n)\Delta_{n-1}(x) + \Delta_{n-2}(x) \\ \Delta_{n-1}(x) &= (x + d_{n-1})\Delta_{n-2}(x) + \Delta_{n-3}(x) \\ &\vdots \end{aligned}$$

$$
\begin{aligned}
\Delta_2(x) &= (x + d_2)\Delta_1(x) + 1 \\
\Delta_1(x) &= (x + d_1)1 + 0.
\end{aligned}
$$

Example 2.7 Given a polynomial $\Delta_5(x) = x^5 + x^3 + 1$, we want to get a $[d_1 d_2 d_3 d_4 d_5]$ of an LHCA with the characteristic polynomial $\Delta_5(x)$. *Assume* that $\Delta_4(x) = x^4 + 1$. Apply the division algorithm five times,

$$
\begin{aligned}
\Delta_5(x) &= x^5 + x^3 + 1 \\
&= (x+0)(x^4+1) + x^3 + x + 1 \\
&= (x + d_5)\Delta_4(x) + \Delta_3(x) \\
\Delta_4(x) &= x^4 + 1 \\
&= (x+0)(x^3 + x + 1) + x^2 + x + 1 \\
&= (x + d_4)\Delta_3(x) + \Delta_2(x) \\
\Delta_3(x) &= x^3 + x + 1 \\
&= (x+1)(x^2 + x + 1) + x \\
&= (x + d_3)\Delta_2(x) + \Delta_1(x) \\
\Delta_2(x) &= x^2 + x + 1 \\
&= (x+1)x + 1 \\
&= (x + d_2)\Delta_1(x) + \Delta_0(x) \\
\Delta_1(x) &= x \\
&= (x+0)1 + 0 \\
&= (x + d_1)\Delta_0(x) + \Delta_{-1}(x) ,
\end{aligned}
$$

we have $[d_1 d_2 d_3 d_4 d_5] = [01100]$ which uniquely defines an LHCA with the characteristic polynomial $\Delta_5(x)$, comparing with Example 2.5 on the page 22. □

The $\Delta_{n-1}(x)$ can be found by solving the equation [12]

$$[\Delta_{n-1}(x)]^2 + (x^2+1)\Delta'_n(x)\Delta_{n-1}(x) + 1 = 0 \bmod \Delta_n(x) , \tag{2.24}$$

where $\Delta'_n(x)$ is the formal derivative of $\Delta_n(x)$, which is calculated in the same manner as the derivative for integer-coefficient polynomials, but with modulo-2. In summary, the Euclidean division algorithm and the equation (2.24) provide a synthesis algorithm which produces LHCA for given irreducible polynomials.

The LHCA corresponding to minimal weight primitive polynomials up to degree 300 can be found in [11]. The minimal weight maximum length LHCA of degrees up to 500 are listed in [13, 40].

2.4 Two-dimensional Linear Cellular Automata

Janoowalla [23] first investigated the structures and behavior of two-dimensional linear cellular automata. In particular, interest was raised in the pseudorandom properties of rectangle-structured LCA (RLCA). Figure 2.6 shows the general structure of degree 4×4 RLCA. Since RLCA with two and three-neighbor cells preserve a comparable hardware cost to that of 1-d LHCA, they are of special interest. The initial statistical and fault simulation results showed that LHCA and the RLCA performed equally well as pseudorandom test pattern generators, and that both are superior to LFSRs.

2.5 Fault Models

A fault is an instance of an incorrect operation of the circuit under test (CUT). The cause of a fault may be fabrication errors, fabrication defects, and physical failures, etc. Some examples of fabrication errors are wrong components, incorrect wiring, and shorts caused by improper solding. Fabrication defects result from an imperfect manufacturing process. Most physical failures occur due to component wear-out and/or environmental factors. The fabrication errors, fabrication defects, and physical failures can collectively be called physical faults

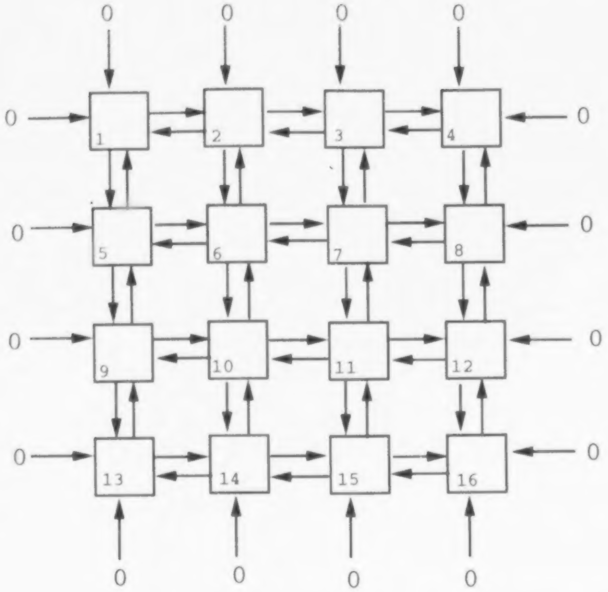


Figure 2.6: The structure of rectangle-structured LCA of degree 4×4

[1, pp. 2-3]. Many different physical faults may cause the same malfunctional effect of the circuit under test. The effect of the physical faults on the circuit's operation is represented by *logical faults*, i.e. logical faults represent the effect of physical faults on the behavior of the modeled system. One approach to abstract the effects of physical faults at some higher level (logic, register transfer, functional block, etc.) is *fault modeling*. If the fault model accurately describes all the physical failures of interest, then one only needs to derive tests to detect all the faults in the fault model.

Fault models have been developed for both gate-level and transistor level descriptions of circuits. The *stuck-at fault model*, which is a gate level fault model, is by far the most prevalent, both in the literature and in the testing industry. Although the stuck-at fault model is widely used, its validity is not universal. Therefore other fault models have been developed which try to account for the physical faults not adequately covered by the stuck-at fault model. Examples of other gate level fault models are the *bridging-fault model* and the *delay fault* *model*. An example of a transistor level fault model is the *stuck-open fault model* in CMOS (complimentary metal oxide semiconductor). In this thesis, stuck-at, stuck-open and delay fault models are considered.

We define a number of terms for later discussion. A *fault set* F for some circuit is defined as a collection of faults under some fault model. A *test pattern* is an input vector which causes an incorrect output in the presence of some specific faults belonging to F. A *test set* T is a set of test patterns. A fault $f \in F$ is said to be detected by a test pattern if that test pattern produces an incorrect output in the presence of f. The test set T is evaluated by determining its quality or effectiveness. The quality of the test set is measured by the ratio of faults it detects to the total number of faults in F. This ratio is called *fault coverage*.

A fault f is said to be *detectable* if there exists a test t that detects f, otherwise, f is an *undetectable* fault. A combinational circuit that contains an undetectable stuck fault is said to be *redundant*, since such a circuit can always be simplified by removing at least one gate or gate input [1, p. 100].

In this work, only permanent faults, that is, the faults that continue to remain in the system after their first occurrence, are considered. The stuck-at, stuck-open and delay fault models are commonly used permanent fault models.

2.5.1 Stuck-at Fault Model

This model assumes that the physical faults in the circuit cause one or more lines to be permanently stuck-at logic 1 or 0. If we assume that there is only a single line in the circuit that is stuck-at 1 or 0, then the model is called the *single stuck-at fault model* (SSF). A single stuck-at fault on a line l stuck-at-α, $\alpha \in \{0, 1\}$, is often denoted by l/α.

Figure 2.7 shows a simple circuit with three inputs; a, b and c, one output; z, and a total of 8 line segments. If line 6 is stuck-at 1, then the line value always remains 1 regardless of the input values a and b. In order to detect this fault by a procedure that allows access only to the primary inputs lines (1, 2 and 3) and the

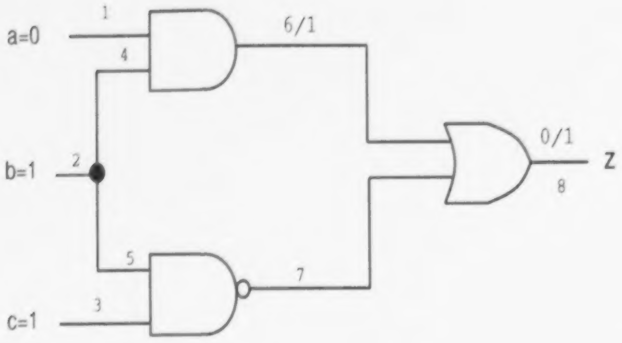


Figure 2.7: Single stuck-at fault model

primary output line (8), it is essential that a test vector must create a change on line 6 and ensure that the change can be seen on line 8. That is, the test vector must produce a 0 on line 6 so that the output created on line 8 clearly shows whether the signal on line 6 is 0 or 1. In the example shown, the output z is equal to 0 for the inputs $a = 0$, $b = 1$, and $c = 1$ assuming no fault in the circuit. However, z is equal to 1 if line 6 is stuck at 1 regardless of the input values. The notation 0/1 in the figure indicates that the fault free value is 0 and the faulty value is 1.

2.5.2 Stuck-open Fault Model

In this model the physical faults convert a combinational CMOS circuit into a sequential one [27, pp. 46-49]. Stuck-open faults are modeled at a transistor level. A stuck-open fault incorporates memory in the circuit, i.e. the circuit retains its previous state. The following example illustrates the effect of a stuck-open fault on a circuit at the transistor level.

Example 2.8 Figure 2.8 shows a CMOS transistor level realization of a 2 input NAND gate. The inputs are labeled a and b and the output is labeled z. The p-type transistors (p_1 and p_2) are connected in parallel, while the n-type transistors (n_1 and n_2) are connected in series. For z to be pulled to V_{DD} (logic 1), at least

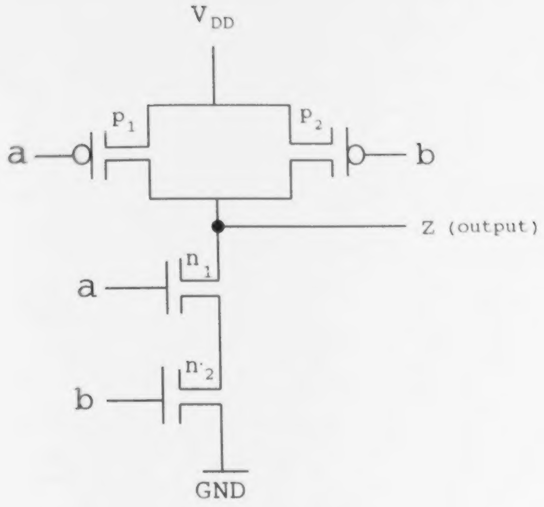


Figure 2.8: CMOS realization of a 2 input NAND gate

inputs		p transistors		n transistors		conducting	output
a	b	p_1	p_2	n_1	n_2	path	z
0	0	ON	ON	OFF	OFF	$V_{DD} \rightarrow z$	1
0	1	ON	OFF	OFF	ON	$V_{DD} \rightarrow z$	1
1	0	OFF	ON	ON	OFF	$V_{DD} \rightarrow z$	1
1	1	OFF	OFF	ON	ON	$z \rightarrow$ GND	0

Table 2.2: Operation of fault free NAND gate

one of the p transistors should have an input of 0. For z to be pulled to GND (logic 0), both n transistors should have an input of 1. Table 2.2 shows the operation of the fault free NAND gate. Assume that there is a break in a line as shown in Figure 2.9. The fault effect is that the transistor p_1 is open, i.e. it cannot conduct. If $a = 0$ and $b = 1$ then the output z is isolated since it is pulled neither to V_{DD} (logic 1) nor to GND (logic 0). Note that transistors p_2, n_1 and n_2 are off and only p_1 is supposed to be on. Since z cannot take the value of V_{DD} or GND, it keeps its present value for at least some length of time due to load capacitance present on the output.

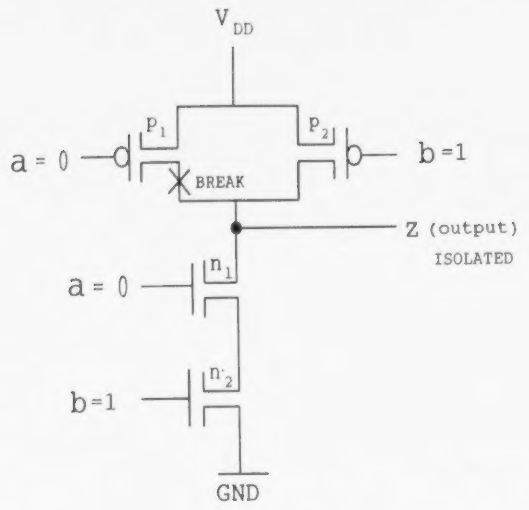


Figure 2.9: CMOS 2 input NAND gate with p_1 open

Two test patterns are required to detect a stuck-open fault: one to set the output line to a certain value, and the next to try and change the value of that output line and observe whether or not it retained its previous value. For the above example, z is first set to 0 ($a = 1$, $b = 1$) and then the pattern $a = 0$, $b = 1$ is applied to change the value of z to 1. Since p_1 is stuck-open, z is isolated and retains 0, thus detecting the fault p_1 stuck-open. It is more difficult to test such faults than faults such as stuck-at since two test patterns are required to detect a stuck-open fault, whereas a stuck-at fault requires a single test pattern.

2.5.3 Transition Fault Model

For the SSF fault model, the *static tests* verify the functional behavior of a given logic circuit under the assumption that the signals are allowed sufficient time to propagate from the inputs of the circuit to its outputs. In the *delay fault model*, it is assumed that some of the signals in the logic may not propagate in time when operating at the system clock speed. The failures causing logic circuits to malfunction at the desired clock rate are called *delay faults* or *AC faults*.

Two types of delay fault models have been mentioned in the literature. The first one is the *transition fault model* (or *gate delay fault model*) [35]. The second is the *path delay fault model* [31]. The path delay fault model considers the cumulative effect of delays along a path from an input to an output. The path delay fault may model a delay defect distributed over a region of a circuit, but the transition fault models a localized fault. In this thesis, the testing of the transition fault model is of interest.

Transition faults are classified into two groups, slow-to-rise and slow-to-fall faults [35]. If a line can change from logic value 0 to 1 but not as fast as it should, then it is said to have a *slow-to-rise* fault. Similarly, a *slow-to-fall* fault is associated with a line if the logic value 1 to 0 transition on the line takes longer than it should.

Let $a_1, a_2, \ldots, a_m$ be the correct(expected) bit sequence for a line in a circuit. Transition faults in that line yield a faulty bit sequence denoted by $y_1, y_2, \ldots y_m$. The slow-to-rise and slow-to-fall faults are defined as follows [35]:

$$\text{slow-to-rise} \;\; y_i = \begin{cases} 1 & \text{if } a_i = 1 \text{ and } a_{i-1} = 1,\ 1 < i \leq m \\ 0 & \text{otherwise} \end{cases}$$

$$\text{slow-to-fall} \;\; y_i = \begin{cases} 0 & \text{if } a_i = 0 \text{ and } a_{i-1} = 0,\ 1 < i \leq m \\ 1 & \text{otherwise} \end{cases}$$

Tables 2.3 and 2.4 show the values for a_{i-1}, a_i, y_i and whether a slow-to-rise or a slow-to-fall fault can be detected, respectively.

If there are n lines in the circuit then there are $2n$ single transition faults [10]. Just as for the stuck-open faults, a pair of test patterns are needed to detect a transition fault: the first sets the line to a_{i-1}, assumed to have a transition fault, and the next sets the line to a_i. That is, the first "sets-up" the fault, and the second "propagates" the fault effect to an output of the circuit.

Current bit a_{i-1}	Expected bit a_i	Faulty bit due to Slow-to-rise delay fault y_i	Fault detected
0	0	0	no
0	1	0	yes
1	0	0	no
1	1	1	no

Table 2.3: Slow-to-rise fault

Current bit a_{i-1}	Expected bit a_i	Faulty bit due to Slow-to-fall delay fault y_i	Fault detected
0	0	0	no
0	1	1	no
1	0	1	yes
1	1	1	no

Table 2.4: Slow-to-fall fault

Chapter 3

Tree-Structured Linear Cellular Automata

It has been shown that two-dimensional rectangular-structured linear cellular automata (RLCA) preserve better test coverage than that of conventional linear feedback shift registers (LFSRs), and are comparable to that of linear hybrid cellular automata (LHCA) [23]. Unlike their one-dimensional counterparts, two-dimensional LCA have many choices of structural configurations. In this chapter, we formally define tree-structured linear cellular automata (TLCA) and explore their cyclic behavior as pseudorandom sequence generators. In particular, low cost maximum-length TLCA are of interest.

3.1 Notations and Definitions

A *tree* consists of a hierarchy of nodes connected by lines. The single node at the top level is the *root* of the tree. A root may have any number of dependent nodes, called its *children*, directly below it. These children nodes, in turn, may have their own children. A node that has no children is called a *leaf*. Two nodes that have the same parent are said to be *siblings*.

The root of the tree is assigned to *level 0*. Its children are on *level 1*. Its children's children are on *level 2*, and so on. The *order* of a tree is the maximum number of children any node has. A *binary tree* is one of order 2, in which each node has no more than two children. Figure 3.1 shows a binary tree.

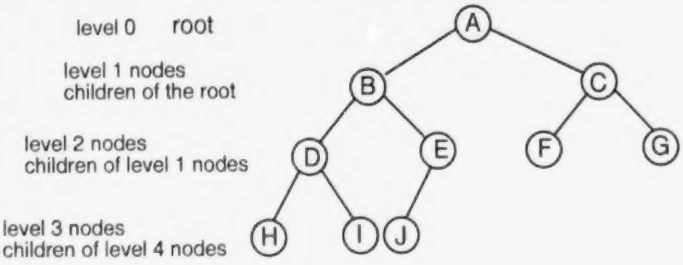

level 0 root
level 1 nodes children of the root
level 2 nodes children of level 1 nodes
level 3 nodes children of level 4 nodes
A
B
C
D
E
F
G
H
I
J

Figure 3.1: A binary tree structure

A *tree-structured linear cellular automata* (TLCA) is a linear finite state machine consisting of a two-dimensional array of storage cells interconnected as a binary tree. A TLCA is capable of performing linear operations over $GF(2)$. The cells of a TLCA are numbered in an ascending order from the root to the leaf, and from the left to the right on each level. The number of cells in a TLCA is defined as the *degree* of the TLCA. Figure 3.2 shows an example of a TLCA. The cell numbers are given at the bottom left corners of the cells. The arrows represent the same meaning as that of LFSRs and LHCA representations, i.e. an in-coming arrow to a cell indicates an input signal and an out-going arrow is an output of the cell.

3.1.1 Computation Rules

Cells in a TLCA perform linear computational rules. However, the numbering convention of the computational rules used in LHCA is inapplicable to TLCA. A cell in a LHCA has 3 possible neighbors (left, right and itself), which permits a total of 2^{2^3} or 256 distinct binary rules (see section 2.3.1). We use the numbers 0 to 255 to represent the rules. The representations are convenient since the numbers are small.

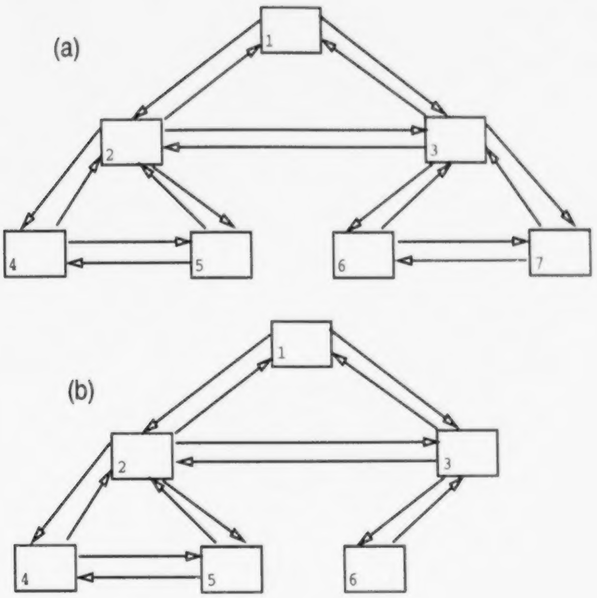


Figure 3.2: (a) A complete TLCA of odd degree 7 (b) A complete TLCA of even degree 6

On the other hand, in a TLCA, there are 5 possible neighbors for each cell. Therefore there are 2^{2^5} or 4,294,967,296 distinct binary rules. The numbers are too large to be used conveniently as rule numbers. Since linear machines are the concern of this research, we consider linear rules only. Let s_i^t be the state of the cell s_i at time t and s_i^{t+1} be the next state of the cell s_i at time $t+1$. The dependency function of the next state of a cell s_i in TLCA is

$$f(s^t_{\lfloor i/2 \rfloor}, s^t_i, s^t_{i+1}, s^t_{2i}, s^t_{2i+1}) \quad = \quad k_1 s^t_{\lfloor i/2 \rfloor} \oplus k_2 s^t_i \oplus k_3 s^t_{i+1} \oplus k_4 s^t_{2i} \oplus k_5 s^t_{2i+1},$$

where a coefficient k_i can be either 0 or 1, and $\lfloor real \rfloor$ is the floor function, which rounds the *real* down to the maximum integer smaller than the *real*.

There exists a total of 2^5 or 32 linear computational rules for TLCA. We number them from 0 to 31. A 5-bit binary string is used to represent the computation rule performed by a TLCA cell. Each bit corresponds to one of the five neighbors. From the most to the least significant bits, they correspond to the

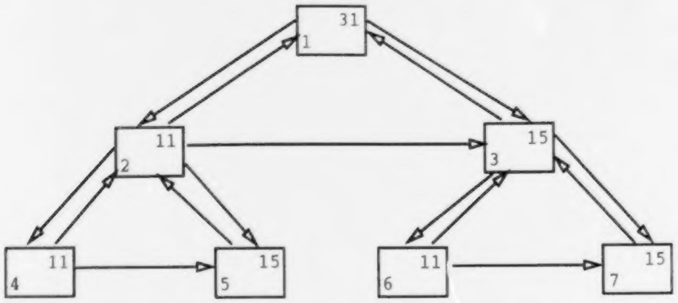


Figure 3.3: A TLCA of degree 7 using rules 31, 23 and 11

self, parent, sibling, left child, and right child of the cell, respectively. A '1' in a bit denotes that the next state of the cell depends on the neighbor corresponding to the bit position, and a '0' denotes no dependency. Similar to that of LHCA, we use the decimal numbers corresponding to the binary strings to name the computation rules. Table 3.1 shows the definitions of all the linear rules.

A root or a leaf TLCA cell does not have the same number of neighbors as the others. Similar to that of LHCA, a *null boundary condition*, i.e. a constant 0 input to a root or a leaf cell, is assumed in this research. The alternative boundary conditions shall be investigated in future research.

We classify the linear rules into two categories: *self-neighboring rules* and *non-self-neighboring rules*. *Self-neighboring rules* encompass rules 16-31, and *non-self-neighboring rules* include rules 0-15. A cell performing a self-neighboring rule is called a *self-neighboring cell*, otherwise, it is called a *non-self-neighboring cell*. The next state of a self-neighboring cell is linearly dependent on its own current state, while that of a non-self-neighboring cell has no such dependency.

Computation rules of a TLCA are closely related to its topological structure. The *structure* of a TLCA represents the interconnections among the cells. The second, third, fourth and fifth bits of the binary representation of a computational rule represent the dependency of the cell of interest on its neighbors which correspond to the interconnections in the structure. The first bit of the rule indicates if the cell is self-neighboring, and can be reflected in the TLCA configuration. A

Self	Parent	Sibling	Left child	Right child	Computation Rule
0	0	0	0	0	0
0	0	0	0	1	1
0	0	0	1	0	2
0	0	0	1	1	3
0	0	1	0	0	4
0	0	1	0	1	5
0	0	1	1	0	6
0	0	1	1	1	7
0	1	0	0	0	8
0	1	0	0	1	9
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18
1	0	0	1	1	19
1	0	1	0	0	20
1	0	1	0	1	21
1	0	1	1	0	22
1	0	1	1	1	23
1	1	0	0	0	24
1	1	0	0	1	25
1	1	0	1	0	26
1	1	0	1	1	27
1	1	1	0	0	28
1	1	1	0	1	29
1	1	1	1	0	30
1	1	1	1	1	31

Table 3.1: TLCA rules

configuration refers to a particular TLCA, where the computational rules of all cells are specified.

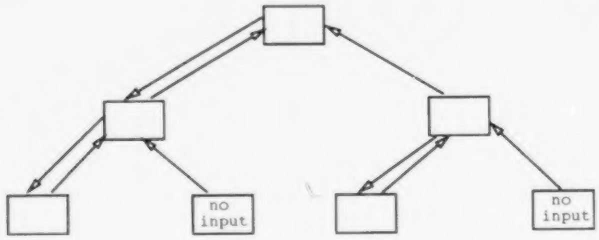


Figure 3.4: An example of a cell in a TLCA without one input

If each cell in a TLCA has inputs from all its neighbors, the structure of the TLCA is *complete*. Otherwise, it is *incomplete*. An n-cell TLCA has one complete and $(4!)^n - 1 = 24^n - 1$ incomplete structures. Figure 3.2 and Figure 3.4 show examples of complete and incomplete TLCA, respectively.

An n-cell TLCA has 24^n structures. A structure has $n!$ configurations. Since each cell in a TLCA can perform any one of the $2^5 = 32$ computation rules, an n-cell TLCA has $24^n \times n!$ possible configurations.

Example 3.1 Figure 3.3 shows a configuration of a degree 7 TLCA, where Rule 31 is used for cell 1, Rule 23 for the other cells with odd numbers, and Rule 11 for the cells with even numbers.

3.1.2 Regularly Structured TLCA

To illustrate the complexity of TLCA structures, let us consider a TLCA of 30 cells. Since each cell can perform any one of the 32 linear computation rules defined, there are $24^{30} \simeq 2.5 \times 10^{41}$ possible structures. In order to reduce the number of the structures, let us consider the rules with two 1's (i.e. the cells with two neighbors) only. A 30-cell TLCA then still has $(C_4^2)^{30} = 6^{30} \simeq 2.2 \times 10^{23}$ possible structures. For a small TLCA as such, the number of possible TLCA structures is already very large.

Definition 3.1 A *TLCA configuration* is *regular* if the all even numbered cells perform either computation rules i or $i + 16$ and the all odd cells perform either computation rules j or $j+16$, where j may or may not equal to i, and $0 \leq i, j \leq 15$.

Definition 3.2 If a *TLCA configuration* is *regular*, then its *structure* is *regular.*

It can be proven that a TLCA has $C_{16}^2 + C_{16}^1 = 136$ regular structures, which is indeed a practical number for consideration. The number is independent of degrees of TLCA. In the rest of this thesis, only TLCA with regular structures and configurations are considered.

The number of possible regular configurations of a n-cell TLCA is

$$n! \times C_{16}^2 \tag{3.1}$$

if the even and odd cells employ different computation rules, and is

$$n! \times 16 \tag{3.2}$$

if the same rule is used for all cells.

3.2 Transition Matrices of TLCA

A $n \times n$ transition matrix, T, is used to represent a TLCA of degree n. The i-th row of the matrix defines the computation rules performed by the i-th cell of the TLCA. For a TLCA cell with k neighbors, where $0 < k \leq 5$, there are exactly k non-zero elements on the corresponding row of the matrix. The element $T_{i,j}$ designates whether the cell i depends on the cell j.

Example 3.2 The TLCA shown in Figure 3.3 has the following next state equations

$$\begin{aligned} s_1^+ &= s_1 \oplus s_2 \oplus s_3 \\ s_2^+ &= s_1 \oplus s_4 \oplus s_5 \end{aligned}$$

$$\begin{aligned} s_3^+ &= s_1 \oplus s_2 \oplus s_6 \oplus s_7 \\ s_4^+ &= s_2 \\ s_5^+ &= s_2 \oplus s_4 \\ s_6^+ &= s_3 \\ s_7^+ &= s_3 \oplus s_7. \end{aligned} \tag{3.3}$$

Therefore its corresponding state transition matrix is

$$T = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3.4}$$

□

Let i and j be the row and column indices of T. T_{ij} can refer to any element in the matrix. In general, the transition matrix T of a TLCA of degree n is characterized by the following notations:

1. an element on the main-diagonal, $T_{i,i}$, here denoted as d_i, indicates if there is a dependency of the cell on itself. If $d_i = 1$, then the next state of the cell i depends on its present state, and if $d_i = 0$ then it does not, where $1 \leq i \leq n$;

2. an element denoted as a_i represents the dependency of the next state of the cell i on the present state of its parent if $a_i = 1$. The element is positioned on the row i and column $\lfloor i/2 \rfloor$, i.e., $T_{i,\lfloor i/2 \rfloor}$, where $2 \leq i \leq n$;

3. an element denoted as b_i states the dependency of the next state of the cell i on its sibling's present state if $b_i = 1$. The element is placed on the row i

and the column k, where

$$\begin{aligned} k &= i+1 \qquad \text{if } i = even \\ k &= i-1 \qquad \text{if } i = odd \ , \end{aligned}$$

i.e., $T_{i,k}$, were $2 \leq i \leq n$;

4. an element c_i denotes the dependency of the next state of the cell i on the present state of its left child if $c_i = 1$. The element is located on the row i and column $2i$, i.e., $T_{i,2i}$, where $1 \leq i \leq n$;
5. an element e_i states the dependency of the next state of the cell i on the present state of its right child if $e_i = 1$. The element is placed on the row i and column $2i + 1$, i.e. $T_{i,2i+1}$, where $1 \leq i \leq n$;
6. all the other elements are 0.

Example 3.3 The transition matrix of an TLCA of degree 7 is

$$T = \begin{pmatrix} d_1 & c_1 & e_1 & 0 & 0 & 0 & 0 \\ a_2 & d_2 & b_2 & c_2 & e_2 & 0 & 0 \\ a_3 & b_3 & d_3 & 0 & 0 & c_3 & e_3 \\ 0 & a_4 & 0 & d_4 & b_4 & 0 & 0 \\ 0 & a_5 & 0 & b_5 & d_5 & 0 & 0 \\ 0 & 0 & a_6 & 0 & 0 & d_6 & b_6 \\ 0 & 0 & a_7 & 0 & 0 & b_6 & d_7 \end{pmatrix} . \tag{3.5}$$

□

3.3 Maximum Length TLCA

In testing applications, pseudorandom sequence generators with maximum length cycle are often required to produce maximum numbers of unique test stimuli, and to exercise as many parts of circuits as possible. Similar to the definition of a maximum length 1-d LHCA or LFSR in the matrix method, the transition matrix of a maximum length TLCA of degree n satisfies the equation

$$T^m = I ,$$

where the smallest value of m is $2^n - 1$. That is, the output sequence of the TLCA has a period of $2^n - 1$. The characteristic polynomial of the transition matrix T of a maximum length TLCA is a primitive polynomial (see Section 2.3.3). Therefore, a *maximum length TLCA* is also called a *primitive TLCA*.

Definition 3.3 A TLCA structure is *primitive* if at least one primitive TLCA configuration of the structure exists at every degree, otherwise, it is *non-primitive.*

In this section, we identify primitive and non-primitive TLCA structures by means of computer simulation, and define the computation rules associated with the structures. The transition matrix of each primitive TLCA structure is defined and illustrated by an example. The implementation cost of the TLCA structures is evaluated.

Exhaustive simulations are performed for the TLCA from degree 2 to 60. This limitation on degrees is due to the complexity of TLCA configuration and the computation power available. For each TLCA, all possible one and two combinations of the 32 computation rules are applied to even and odd cells, and the primitivity test is conducted using Algorithm I described in Section 4.1. It has been found that certain TLCA structures do not have maximum length machines. A comprehensive exposition of primitive TLCA structures will be given in Section 3.3.2.

3.3.1 Non-primitive TLCA Structures

In Section 3.1.2, it is given that the number of the possible regularly structured TLCA is 136. The results of our simulations show that most TLCA structures define non-primitive machines. Only five primitive TLCA structures are found from degree 2 to 60.

We intend to characterize non-primitive structures from primitive ones. It is easy to verify that a TLCA structure is non-primitive if a cell in the TLCA does not have any input from the other cells. This criteria rejects a large number of non-primitive TLCA. Figure 3.4 shows an example of a degree 7 non-primitive TLCA structure. However, it is also found that a TLCA structure can be non-primitive even if every cell has at least one input from the others. Figure 3.5 depicts some examples of non-primitive TLCA structures.

3.3.2 Primitive TLCA Structures

Five TLCA structures with maximal length cycles have been found. For convenience, we use TLCA(I) to denote the Type I TLCA structure, TLCA(II) for the Type II TLCA structure, and so on.

Our exhaustive simulation results show that from degree 2 to 60:

1. all the five primitive TLCA structures are incomplete;
2. certain computation rules used in certain TLCA structures result in primitive machines;
3. cells of a TCLA sharing an identical computation rule do not result in primitive machines.

Table 3.2 summarizes the self- and non-self-neighboring rules which can be used in the five primitive TLCA structures. The rule numbers without parentheses are non-self-neighboring rules and the ones in parentheses are self-neighboring rules. A pair of rules in an entry indicates that either can be used for the cell type of a TLCA structure. This is implies that the maximum number of rules that the cells of a TLCA structure can have is four. A TLCA may have three or two different computation rules. Slight modifications are required for TLCA(I) and TLCA(IV) of even degrees. We will discuss it later in this section.

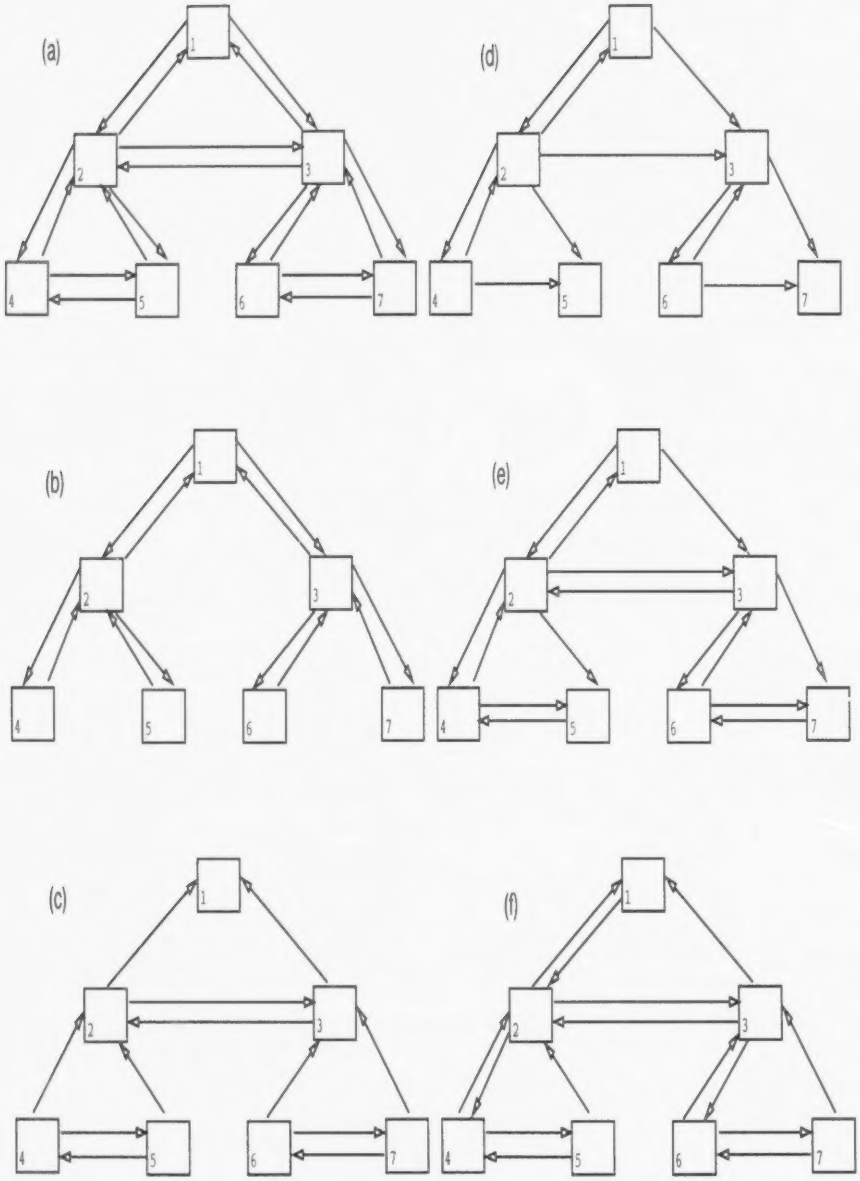


Figure 3.5: Examples of non-primitive TLCA structures

Cells in	TLCA(I)	TLCA(II)	TLCA(III)	TLCA(IV)	TLCA(V)
even numbers	9(25)	11(27)	11(27)	6(22)	14(30)
odd numbers	5(21)	7(23)	15(31)	14(30)	10(26)

Table 3.2: Computation rules of five primitive TLCA structures

The five primitive TLCA structures are depicted in Figures 3.6, 3.7, 3.8, 3.9 and 3.10. The number at the top right corner of each cell is the computational rule using the same convention as explained for Table 3.2.

We describe the detailed structures and configurations of the five primitive TLCA, and provide examples using degree 6 and 7 TLCA.

1. The TLCA(I) structure shown in Figure 3.6 (a) uses *rule 9 (or rule 25)* for the cells with even numbers, and *rule 5 (or rule 21)* for the cells with odd numbers. However, if a degree n is even, this structure needs to be slightly modified, i.e. the parent cell of the last cell of the TLCA employs *rule 6 (or rule 22)*. The modification rule is as follows. In Figure 3.6 (a), the cells 3, 6 and 7 form a circle. If the cell 7 is removed to construct a degree 6 TLCA, it would break the circular connections between the cells 3 and 6. The modification of the computation rule for the cell 3 retains the primitivity of TLCA(I) of even degrees. The modification rule is applicable to TLCA(I) of higher degrees and TLCA(IV). Figure 3.6 (b) demonstrates the structure of a TLCA(I) of degree 6.

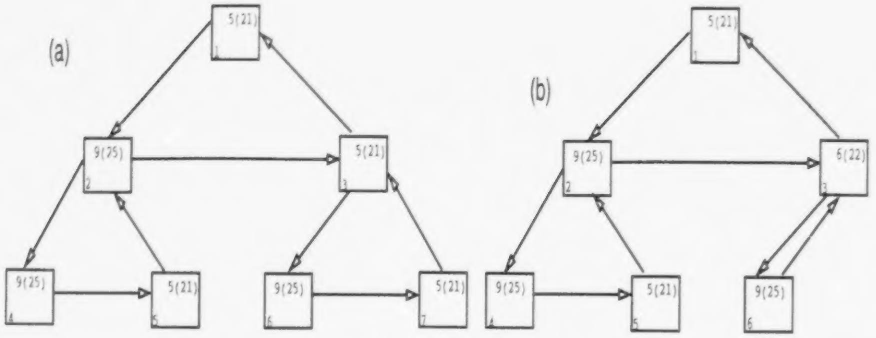


Figure 3.6: (a) TLCA(I) structure of odd degree (b) TLCA(I) of even degree

The non-zero elements of the transition matrix of the TLCA(I) are

$$\begin{array}{ll} (1) & \left.\begin{array}{r} a_i = 1 \\ b_i = 0 \\ c_i = 0 \\ e_i = 1 \end{array}\right\} \quad \text{if row } i = even \\ (2) & \left.\begin{array}{r} a_i = 0 \\ b_i = 1 \\ c_i = 0 \\ e_i = 1 \end{array}\right\} \quad \text{if row } i = odd \end{array} \tag{3.6}$$

Example 3.4 The transition matrix of TLCA(I) of degree 7 (an odd degree) is

$$T = \begin{pmatrix} d_1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & d_2 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & d_3 & 0 & 0 & \underline{0} & 1 \\ 0 & 1 & 0 & d_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & d_5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & d_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & d_7 \end{pmatrix}, \tag{3.7}$$

and the transition matrix of TLCA(I) of degree 6 (an even degree) is

$$T = \begin{pmatrix} d_1 & 0 & 1 & 0 & 0 & 0 \\ 1 & d_2 & 0 & 0 & 1 & 0 \\ 0 & 1 & d_3 & 0 & 0 & \underline{1} \\ 0 & 1 & 0 & d_4 & 0 & 0 \\ 0 & 0 & 0 & 1 & d_5 & 0 \\ 0 & 0 & 1 & 0 & 0 & d_6 \end{pmatrix}. \tag{3.8}$$

Note that the matrix T of degree 6 is exactly the same as the sub-matrix of the transition matrix T of degree 7, except for the difference in the elements on the row 3 and the column 6, i.e. the underlined elements. They reflect the difference in the computational rules before and after the modification of the parent cell of the last leaf cell. □

2. The TLCA(II) structure shown in Figure 3.7 uses *rule 11 (or rule 27)* for the cells with even numbers, and *rule 7 (or rule 23)* for other cells with odd numbers. Note that since a circular connection exists between the cells 3 and 6 of a TLCA(II) of degree 6, the TLCA(II) does not require a modification of cell 3.

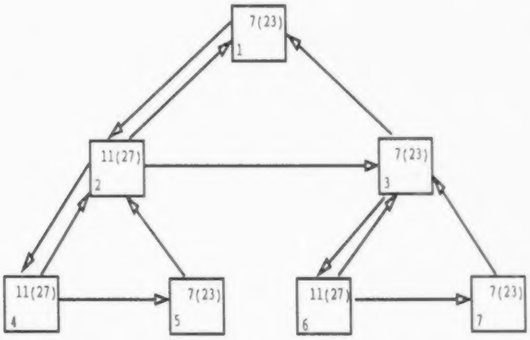


Figure 3.7: TLCA(II) structure

The non-zero elements of the transition matrix of the TLCA are

$$\begin{array}{ll}(1) & \left.\begin{array}{l} a_i = 1 \\ b_i = 0 \\ c_i = 1 \\ e_i = 1 \end{array}\right\} \quad \text{if row } i = \mathit{even} \\ (2) & \left.\begin{array}{l} a_i = 0 \\ b_i = 1 \\ c_i = 1 \\ e_i = 1 \end{array}\right\} \quad \text{if row } i = \mathit{odd} \end{array} \tag{3.9}$$

Example 3.5 The transition matrix of TLCA(II) of degree 7 is

$$T = \begin{pmatrix} d_1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & d_2 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & d_3 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & d_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & d_5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & d_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & d_7 \end{pmatrix}. \tag{3.10}$$

□

3. The TLCA(III) structure shown in Figure 3.8 uses *rule 11 (or rule 27)* for the cells with even numbers, and *rule 15 (or rule 31)* for other cells with odd numbers.

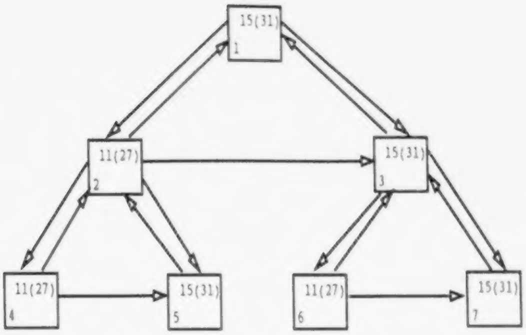


Figure 3.8: TLCA(III) structure

The non-zero elements of the transition matrix of the TLCA are

$$\begin{array}{lll} (1) & \left.\begin{array}{l} a_i = 1 \\ b_i = 0 \\ c_i = 1 \\ e_i = 1 \end{array}\right\} & \text{if row } i = \mathit{even} \\ (2) & \left.\begin{array}{l} a_i = 1 \\ b_i = 1 \\ c_i = 1 \\ e_i = 1 \end{array}\right\} & \text{if row } i = \mathit{odd} \end{array} \tag{3.11}$$

Example 3.6 The transition matrix of TLCA(III) of degree 7 is

$$T = \begin{pmatrix} d_1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & d_2 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & d_3 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & d_4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & d_5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & d_6 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & d_7 \end{pmatrix}. \tag{3.12}$$

□

4. The TLCA(IV) structure shown in Figure 3.9 (a) uses *rule 6 (or rule 22)* for the cells with even numbers, and *rule 14 (or rule 30)* for the cells with odd numbers. But for a TLCA(IV) of even degree, the last cell of the TLCA is modified to use *rule 14 (or rule 30)* in order to retain the circular structure between the cells 3 and 6. Figure 3.9 (b) illustrates a TLCA(IV) of degree 6.

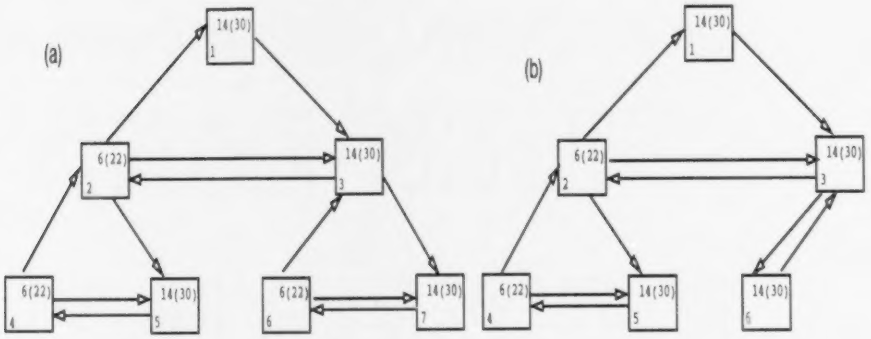


Figure 3.9: (a) TLCA(IV) structure of odd degree (b) TLCA(IV) of even degree

The non-zero elements of the transition matrix of the TLCA are

$$\begin{array}{cl} (1) & \left.\begin{array}{l} a_i = 0 \\ b_i = 1 \\ c_i = 1 \\ e_i = 0 \end{array}\right\} \quad \text{if row } i = \textit{even} \\ (2) & \left.\begin{array}{l} a_i = 1 \\ b_i = 1 \\ c_i = 1 \\ e_i = 0 \end{array}\right\} \quad \text{if row } i = \textit{odd} \end{array} \tag{3.13}$$

Example 3.7 The transition matrix of TLCA(IV) of degree 7 is

$$T = \begin{pmatrix} d_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & d_3 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & d_4 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & d_5 & 0 & 0 \\ 0 & 0 & \underline{0} & 0 & 0 & d_6 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & d_7 \end{pmatrix}, \qquad (3.14)$$

and the transition matrix of TLCA(IV) of degree 6 is

$$T = \begin{pmatrix} d_1 & 1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 1 & 1 & 0 & 0 \\ 1 & 1 & d_3 & 0 & 0 & 1 \\ 0 & 0 & 0 & d_4 & 1 & 0 \\ 0 & 1 & 0 & 1 & d_5 & 0 \\ 0 & 0 & \underline{1} & 0 & 0 & d_6 \end{pmatrix}. \qquad (3.15)$$

□

5. The TLCA(V) structure shown in Figure 3.10 uses *rule 14 (or rule 30)* for the cells with even numbers, and *rule 10 (or rule 26)* for other cells with odd numbers.

The non-zero elements of the transition matrix of the TLCA are

$$\begin{array}{ll} (1) & \left.\begin{array}{l} a_i = 1 \\ b_i = 1 \\ c_i = 1 \\ e_i = 0 \end{array}\right\} \quad \text{if row } i = even \\ (2) & \left.\begin{array}{l} a_i = 1 \\ b_i = 0 \\ c_i = 1 \\ e_i = 0 \end{array}\right\} \quad \text{if row } i = odd \end{array} \qquad (3.16)$$

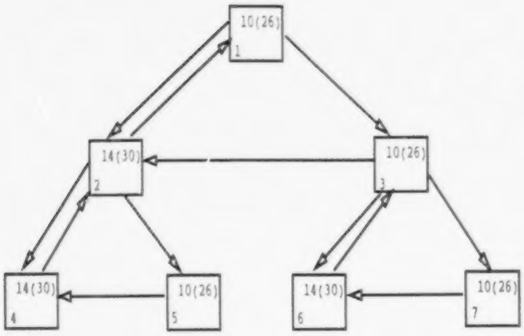


Figure 3.10: TLCA(V) structure

Example 3.8 The transition matrix of TLCA(V) of degree 7 is

$$T = \begin{pmatrix} d_1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & d_2 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & d_3 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & d_4 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & d_5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & d_6 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & d_7 \end{pmatrix}. \tag{3.17}$$

□

3.3.3 Hardware Cost of TLCA Structures

Hardware cost of pseudorandom sequence generators (PSG) is an important issue in BIST design and implementation. The exact hardware cost of a PSG can be obtained by implementing the machine using a computer-aided design (CAD) tool. However, the result of this estimation depends on many factors of the implementation of the CAD tool used, such as the placement and routing algorithms, and the implementation of the cell libraries. Here, we are interested in obtaining a rough estimation of the hardware cost of TLCA in order to compare the results with that of LHCA. We use a gate counting method, which is independent of CAD tools and implementation technology.

A LHCA or TLCA cell consists of a storage element and some XOR gates to perform required computation. Since a storage element is required for both LHCA and TLCA, the hardware cost can be evaluated only on the cost of XOR gates. Given a TLCA defined by the major diagonal elements of its transition matrix and its structure type, the number of XOR gates required is calculated using the following rules: a '1' major diagonal element indicates a self-neighboring cell, i.e. having three neighbors, including itself, thus, requires two 2-input XOR gates; a '0' major diagonal element represents a non-self-neighboring cell requiring one 2-input XOR gate.

Consider LHCA and TLCA with non-self-neighboring cells only, and let n be the degree of the machines. A 1-d LHCA requires $(n-2)$ 2-input XOR gates. A TLCA(I) needs

$$\begin{array}{ll} \dfrac{n-2}{2} & \text{if } n = even \\ \dfrac{n-3}{2} & \text{if } n = odd \end{array} \tag{3.18}$$

2-input XOR gates. A TLCA of a Type II, IV or V structure shares the same number of XOR gates as that of 1-d LHCA, $n-2$. TLCA(III) is the most expensive type, which requires

$$\begin{array}{ll} 3 \cdot \frac{n-3}{2} + 2 & \text{if } n = odd \\ 3 \cdot \frac{n-2}{2} & \text{if } n = even \end{array} \tag{3.19}$$

XOR gates.

Table 3.3 summarizes the number of XOR gates required for machines of degrees up to 15. It can be seen that TLCA(I) is the most cost effective structure, which permits about 50% fewer XOR gates than that of LHCA. In other words, its cost is closer to that of commonly used LFSRs. TLCA(II), (IV) and (V) have the same cost as 1-d LHCA. TLCA (III), however, requires almost 1.5 times the number of XOR gates of 1-d LHCA.

Degree	LHCA	TLCA(I)	TLCA(II)	TLCA(III)	TLCA(IV)	TLCA(V)
2	0	0	0	0	0	0
3	1	0	1	2	1	1
4	2	1	2	3	2	3
5	3	1	3	5	3	3
6	4	2	4	6	4	4
7	5	2	5	8	5	5
8	6	3	6	9	6	6
9	7	3	7	11	7	7
10	8	4	8	12	8	8
11	9	4	9	14	9	9
12	10	5	10	15	10	10
13	11	5	11	17	11	11
14	12	6	12	18	12	12
15	13	6	13	20	13	13

Table 3.3: Number of XORs in LHCA and Types I to V TLCA

3.4 Summary

In this chapter, we formally introduced tree-structured linear cellular automata (TLCA), and defined their computation rules and transition matrices. The five types of maximum length TLCA were found. The hardware cost of the maximum length TLCA were evaluated in comparison to 1-d LHCA. The TLCA(I) is the most cost effective implementation among the 1-d and 2-d linear cellular automata. In following chapters, we will examine the pseudorandom behavior of TLCA, and their potentials as pseudorandom sequence generators in testing applications.

Chapter 4

Maximum Length TLCA

Linear finite state machines (LFSMs) with maximum length cycle have been found to be particularly useful in engineering applications. To determine if a LFSM has the maximum length cycle is a computationally intensive task. A common practice today is to use lookup tables that contain lists of LFSMs with desired properties, provided by researchers and scientists.

In the previous chapter, we defined the structures, configurations and operations of tree-structured linear cellular automata (TLCA). In the rest of this thesis, the term TLCA refers to a configuration of a TLCA structure. In this chapter, we present two computational algorithms that determine maximum length TLCA. Special consideration is given to the implementation of the algorithms, due to their computational complexity. Lookup tables of low cost maximum length TLCA are provided up to degree 60.

4.1 Algorithm I

Similar to one-dimensional LFSMs, LFSRs and LHCA, a length n TLCA is defined by a $n \times n$ transition matrix. The mapping between characteristic polynomials and transition matrices is one-to-one. Therefore, the polynomial sieve methods given in Section 2.2.3 can be adopted to determine the primitivity of a given

TLCA. The sieve algorithm proposed here is based on matrix multiplication and is called Algorithm I. Conceptually, it consists of two tests: irreducibility and primitivity tests. Only the TLCA that passes the irreducibility test are tested for their primitivity.

Recall in Section 2.2.2, an $n \times n$ transition matrix T of a length n LFSR satisfies the equation

$$T^{(2^i-1)} = I, \qquad or \qquad T^{2^i} = T,$$

where $i \leq n$, and I is the identity matrix. The period of the output sequence defined by the equations is $(2^i - 1)$. If $i = n$, the characteristic polynomial of the matrix T is irreducible; otherwise, it is reducible. Since the irreducibility of a polynomial is a necessary condition for primitivity, a reducible polynomial is non-primitive, thus, it does not have the maximum length cycle.

For a TLCA under test, the detailed algorithm is as follows:

1. compute all prime cofactors[1] of $2^n - 1$, add 1 to each number obtained, and store them in a table F;
2. construct the transition matrix T of the TLCA following the equations given in Section 3.3.2;
3. irreducibility test:
    (1) compute T^{2^i}, for $i = 1, 2, \cdots, n-1$, using the powering algorithm described in Section 4.2.4. If the equation $T^{2^i} = T$ holds for any i, reject the TLCA, go to step 5; otherwise, go to step 3 (2);

    (2) compute T^{2^n}. If $T^{2^n} = T$, the TLCA is irreducible, go to step 4; otherwise reject the TLCA, go to step 5;

---

[1] If p is a prime factor of an integer q, i.e. $p|q$ and p is prime, then $k = q/p$ is a *prime cofactor* of q.

4. primitivity test: compute T^q for each element q in the table F, again using the powering algorithm. If $T^q = T$ for any q, reject the TLCA, go to step 5; otherwise the TLCA is primitive, go to step 5;

5. the end of test.

The primitivity test is non-deterministic since it depends on both the number of prime factors of $2^n - 1$ and which prime factor in the table rejects the TLCA under consideration. In contrast, the computational complexity of the irreducibility test can be determined.

An element of the product of two matrices, A and B, is calculated by

$$\sum_{m=1}^{n} a_{im} \cdot b_{mj}. \tag{4.1}$$

Assuming that $a_{im} \cdot b_{mj}$ takes a constant CPU time, c, the computation time required by Equation 4.1 is $c\,n$. The time required by the entire matrix multiplication is $cn \cdot n \cdot n = cn^3$. To compute T^{2^n}, the CPU time required is $n \cdot cn^3 = cn^4$ (see Section 4.2.4). Thus, the computational complexity of the irreducibility test is $O(n^4)$.

In Figure 4.1 the stars show the actual CPU time taken to test the irreducibility of a given matrix as a function of degrees of TLCA on a SUN SPARC-20 workstation. The solid line represents the CPU times predicted by the computational complexity analysis given above, where $c = 6.31293232 \times 10^{-7}$ seconds is used in the calculation.

4.2 Implementation of Algorithm I

As stated in the previous section, Algorithm I is computationally intensive. When the implementation of the algorithm is concerned, there are two main problems: the system limits of integer representation, and the CPU speed of a computer system.

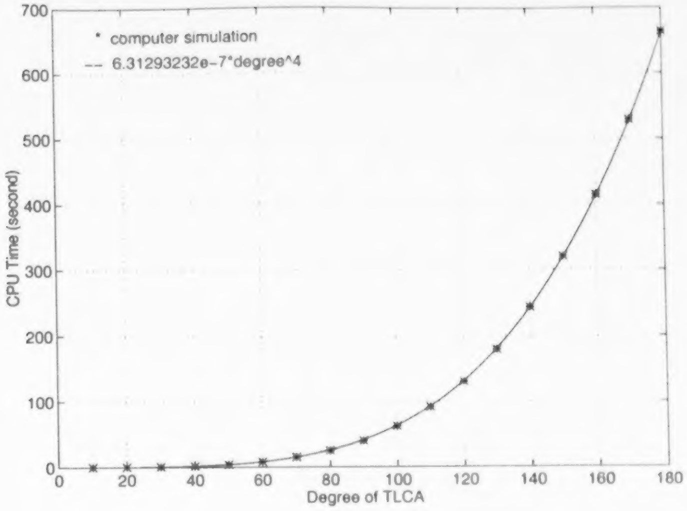


Figure 4.1: Computational complexity ($O(n^4)$) of the irreducibility test

For the former, consider a degree 60 polynomial that defines a TLCA of degree 60 and a commonly used 32-bit computer. To test the primitivity of the TLCA, we need to find all the prime factors of $2^{60}-1$ ($= 1,152,921,504,606,846,975$), which far exceeds the system limit on the maximum integer, typically, $2^{31}-1 = 2,147,483,647$.

For the latter, it is not uncommon in engineering practice that LFSMs over degree 100 are required. LFSMs of degree two to three hundred are desirable in some testing applications. When running Algorithm I for high degree polynomials, the speed of a computer system is crucial. This section is devoted to the special treatments of implementation issues of the algorithm. The complete C code of the implementation can be found in Appendix B.

4.2.1 Multiple Precision Integers

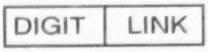
A solution to representing integers that exceed the system limits of a computer system is to impose a data structure on the basic data type, *single-precision integer* [18, 29]. The data structure is capable of representing a *multiprecision integer*

d (or *multiple precision integer*). The integer d is partitioned into l segments $(d_0, d_1, \cdots, d_{l-1})$ of single-precision integers. The least significant segment is d_0, and the most significant segment is d_{l-1}. Then, the multiprecision integer can be represented by

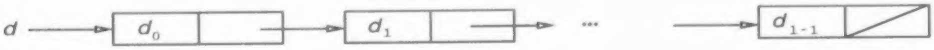
$$d = \sum_{i=0}^{l-1} d_i \beta^i \ , \tag{4.2}$$

where β is the base, i.e. the maximum value represented by d_i. β may be any positive integer between 2 and the upper limit of a *single-precision integer.*

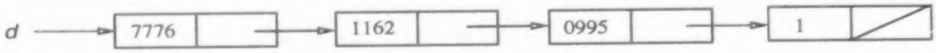
The data structure can be implemented by a linked list where each node in the linked list is of the form



The field DIGIT contains a base-β integer and the field LINK consists of a pointer pointed to the next node of the linked list, or a NULL pointer indicating the end of the list. Thus the equation (4.2) can be represented by the linked list



Example 4.1 Let $\beta = 10^4$. Partition the decimal number $2^{40} = 1099511627776$ into four segments, 1 0995 1162 7776. Then the linked list representing the number is



□

The algorithms performing the *multiprecision integer* addition and multiplication can be found in [29]. In our implementation, the GNU Multiple Precision Arithmetic Library [20] is used to process multiprecision integers.

4.2.2 Prime Factors of $2^n - 1$

To test primitivity of a given transition matrix of a TLCA of degree n, the prime factors of $2^n - 1$ are needed. Since the primitivity test of TLCA of the same degree share the same set of prime factors, we precompute the factors and store them in a look-up table in order to avoid the repeat calculation and reduce the computation time. One should be aware of the fact that the calculation of prime factors is extremely CPU-intensive. It may take months for the calculations of large degree. The table of prime factors are partly obtained by using Maple [14] and PARI [6], and partly from the report of Cunningham Project [9].

4.2.3 Matrix Multiplication

In Algorithm I, matrix multiplication (in binary field) is the most extensively used operation. Therefore, the efficiency of its implementation has a significant impact on the efficiency of the algorithm.

We use integer comparisons and additions in the following pseudo-code

```
c[i][j] := 0;         {* two-dimensional array *}
for m := 1 to n do
   begin
      if ( a[i][m] = 1 )
          c[i][j] := ( c[i][j] + b[m][j] ) mod 2;
   end
```

to perform the product of matrices A and B

$$c_{ij} = \sum_{m=1}^{n} a_{im} \cdot b_{mj} \, .$$

It has been shown that the implementation of the pseudo-code is at least two times faster than that of the direct multiplication on SPARCstations.

4.2.4 Powering Algorithm

The powering algorithm [15] is used in both the irreducibility and primitivity tests of Algorithm I. It provides an efficient implementation of

$$a^k \bmod p \tag{4.3}$$

for an integer k. If we write the integer k in binary as

$$k = \sum_{i=0}^{\lfloor \log_2 k \rfloor} b_i \cdot 2^i \tag{4.4}$$

where $b_i \in \{0, 1\}$, and $\lfloor real \rfloor$ is the floor function, which is the maximum integer smaller than the *real*, then a^k can be computed by

$$a^k = \prod_{i=0}^{\lfloor \log_2 k \rfloor} a^{b_i \cdot 2^i}.$$

Therefore, the modular operation (4.3) can be efficiently accomplished by repeatly squaring ($a^{2^i} = (a^{2^{i-1}})^2$).

Example 4.2 To calculate 7^{11} mod 17, we first calculate

$$7^2 \bmod 17 = 15, \quad 7^4 \bmod 17 = 15^2 = 4, \quad 7^8 \bmod 17 = 4^2 = 16\ .$$

Since $7^{11} = 7 \cdot 7^2 \cdot 7^8$, 7^{11} mod 17 equals to the product of the remainders of 7^2, 7^4 and 7^8 mod17, i.e.

$$7^{11} = 7 \cdot 15 \cdot 16 = 14\ .$$

□

The complexity of the powering algorithm is $O(\log_2 k)$, as opposed to $O(k)$ for the direct computation.

4.3 Algorithm II

Algorithm I presented in the previous section is a sieve method based on matrix multiplication. Algorithm II that we propose here uses a more direct approach. It derives recurrence relations of d_1, d_2, $\cdots$, and d_n to form the characteristic polynomial of the transition matrix of a TLCA. The principles of the algorithm are derived from Definition 2.10 and Theorems 2.4 and 2.5.

Algorithm II is given as follows:

1. Form the transition matrix of a TLCA;
2. Compute the characteristic polynomial of the transition matrix by using recurrence relations;
3. Check if the characteristic polynomial is primitive. If the polynomial is primitive then a maximum length machine has been found.

The recurrence relations can be determined by using Laplace Expansion of a determinant. Let $\Delta_k(x)$ be the characteristic polynomial of degree k. For a transition matrix of TLCA(III) described in Section 3.3.2, we have the following recurrence relations, similar to Equation (2.23), to obtain its characteristic polynomial:

$$\Delta_k(x) = g_k \Delta_{k-1}(x) + Q_{k-1}, \qquad k = even \tag{4.5}$$

and

$$\Delta_{k+1}(x) = g_{k+1} \Delta_k(x) + Q_{k-1}(g_k + 1)\ , \tag{4.6}$$

with the initial conditions $\Delta_{-1}(x) = 0$ and $\Delta_0(x) = 1$, where $g_k = x + d_k$ and Q_{k-1} is the quotient of

$$\Delta_{k-1}(x) = g_{k/2} \cdot Q_{k-1}(x) + R_{k-1}\ , \tag{4.7}$$

where R_{k-1} is the reminder.

The proof of the equations (4.5), (4.6), and (4.7) requires the induction method, which is tedious. Here we give two examples to verify them.

Example 4.3 The transition matrix of a maximum length TLCA(III) [1100] is

$$T = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} .$$

Its characteristic polynomial is computed as follows.

$$\begin{aligned}
\Delta_{-1}(x) &= 0 \\
\Delta_0(x) &= 1 \\
&\quad \Rightarrow Q_{-1} = 0 \\
\Delta_1(x) &= g_1\Delta_0 + Q_{-1}(g_{-1} + 1) \\
&= g_1 \\
&\quad \Rightarrow Q_1 = 1 \\
\Delta_2 &= g_2\Delta_1 + Q_1 \\
&= g_2g_1 + 1 \\
\Delta_3 &= g_3\Delta_2 + Q_1(g_2 + 1) \\
&= g_3g_2g_1 + g_3 + g_2 + 1 \\
&\quad \Rightarrow Q_3 = g_3g_1 + 1 \\
\Delta_4 &= g_4\Delta_3 + Q_3 \\
&= g_4g_3g_2g_1 + g_4g_3 + g_4g_2 + g_4 + g_3g_1 + 1 .
\end{aligned}$$

After replacing $g_1 = x + 1$, $g_2 = x + 1$, $g_3 = x$ in Δ_4, and $g_4 = x$, and simplifying it, we have the characteristic polynomial

$$\Delta_4 = x^4 + x + 1 .$$

The look-up tables of TLCA and their corresponding characteristic polynomials are given in Section 5.4 from page 79 to 81. □

Example 4.4 The characteristic polynomial of transition matrix of TLCA(III) of degree 6 is

$$\begin{aligned}
\Delta_6 &= \begin{vmatrix} g_1 & 1 & 1 & 0 & 0 & 0 \\ 1 & g_2 & 0 & 1 & 1 & 0 \\ 1 & 1 & g_3 & 0 & 0 & 1 \\ 0 & 1 & 0 & g_4 & 0 & 0 \\ 0 & 1 & 0 & 1 & g_5 & 0 \\ 0 & 0 & 1 & 0 & 0 & g_6 \end{vmatrix} \\
&= g_6\Delta_5 + \begin{vmatrix} g_1 & 1 & 1 & 0 & 0 \\ 1 & g_2 & 0 & 1 & 1 \\ 0 & 1 & 0 & g_4 & 0 \\ 0 & 1 & 0 & 1 & g_5 \\ 0 & 0 & 1 & 0 & 0 \end{vmatrix} \\
&= g_6\Delta_5 + \begin{vmatrix} g_1 & 1 & 0 & 0 \\ 1 & g_2 & 1 & 1 \\ 0 & 1 & g_4 & 0 \\ 0 & 1 & 1 & g_5 \end{vmatrix} \\
&= g_6\Delta_5 + Q_5
\end{aligned}$$

and the characteristic polynomial of the transition matrix of TLCA(III) of degree 7 is

$$\Delta_7 = \begin{vmatrix} g_1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & g_2 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & g_3 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & g_4 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & g_5 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & g_6 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & g_7 \end{vmatrix}$$

$$
\begin{aligned}
&= g_7\Delta_6 + \begin{vmatrix} g_1 & 1 & 1 & 0 & 0 & 0 \\ 1 & g_2 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & g_4 & 0 & 0 \\ 0 & 1 & 0 & 1 & g_5 & 0 \\ 0 & 0 & 1 & 0 & 0 & g_6 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{vmatrix} \\
&= g_7\Delta_6 + \begin{vmatrix} g_1 & 1 & 0 & 0 & 0 & 1 \\ 1 & g_2 & 1 & 1 & 0 & 0 \\ 0 & 1 & g_4 & 0 & 0 & 0 \\ 0 & 1 & 1 & g_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & g_6 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{vmatrix} \\
&= g_7\Delta_6 + \begin{vmatrix} g_1 & 1 & 0 & 0 \\ 1 & g_2 & 1 & 1 \\ 0 & 1 & g_4 & 0 \\ 0 & 1 & 1 & g_5 \end{vmatrix} \begin{vmatrix} g_6 & 1 \\ 1 & 1 \end{vmatrix} \\
&= g_7\Delta_6 + Q_5(g_6+1),
\end{aligned}
$$

but the characteristic polynomial of the transition matrix of TLCA(III) of degree 5 is

$$
\begin{aligned}
\Delta_5 &= \begin{vmatrix} g_1 & 1 & 1 & 0 & 0 \\ 1 & g_2 & 0 & 1 & 1 \\ 1 & 1 & g_3 & 0 & 0 \\ 0 & 1 & 0 & g_4 & 0 \\ 0 & 1 & 0 & 1 & g_5 \end{vmatrix} \\
&= g_3 \begin{vmatrix} g_1 & 1 & 0 & 0 \\ 1 & g_2 & 1 & 1 \\ 0 & 1 & g_4 & 0 \\ 0 & 1 & 1 & g_5 \end{vmatrix} + \begin{vmatrix} 1 & g_2 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & g_4 & 0 \\ 0 & 1 & 1 & g_5 \end{vmatrix}
\end{aligned}
$$

$$= g_3 Q_5 + R_5$$

where the quotient and reminder are

$$Q_5 = \begin{vmatrix} g_1 & 1 & 0 & 0 \\ 1 & g_2 & 1 & 1 \\ 0 & 1 & g_4 & 0 \\ 0 & 1 & 1 & g_5 \end{vmatrix}$$

$$R_5 = \begin{vmatrix} 1 & g_2 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & g_4 & 0 \\ 0 & 1 & 1 & g_5 \end{vmatrix}$$

respectively. □

Similarly, we have derived the recurrence relations for:

1. TLCA(I)

$$\begin{aligned} \Delta_k(x) &= g_k\Delta_{k-1}(x) + Q_{k-1}, \qquad k = even \qquad (4.8) \\ \Delta_{k+1}(x) &= g_{k+1}g_k\Delta_{k-1}(x) + Q_{k-1} \\ \Delta_{k-1}(x) &= g_{k/2} \cdot Q_{k-1}(x) + R_{k-1} \end{aligned}$$

2. TLCA(IV)

$$\begin{aligned} \Delta_k(x) &= g_k\Delta_{k-1}(x) + Q_{k-1}, \qquad k = even \qquad (4.9) \\ \Delta_{k+1}(x) &= (g_{k+1}g_k + 1)\Delta_{k-1}(x) + Q_{k-1} \\ \Delta_{k-1}(x) &= g_{k/2} \cdot Q_{k-1}(x) + R_{k-1} \end{aligned}$$

3. TLCA(II) and TLCA(V)

$$\begin{aligned} \Delta_k(x) &= g_k\Delta_{k-1}(x) + Q_{k-1}, \qquad k = even \qquad (4.10) \\ \Delta_{k+1}(x) &= g_{k+1}\Delta_k(x) + Q_{k-1} \\ \Delta_{k-1}(x) &= g_{k/2} \cdot Q_{k-1}(x) + R_{k-1}. \end{aligned}$$

The algorithm has been implemented in Maple [14], a powerful symbolic computation package. Since the algorithm requires many polynomial computations using the recurrence relations, the implementation is very computational intensive. The current version of Maple is inefficient for multi-variable polynomial multiplication. This makes the algorithm impractical for TLCA when degree $n \geq 25$ for the time being.

4.4 Minimal Cost Maximum Length TLCA

As stated in Section 3.3.3, LFSMs of all types require one storage device per cell. Therefore, the hardware cost of TLCA can be measured by the number of XOR gates required in their implementation. Maximum length TLCA with a minimum number of 1's corresponds to the most cost effective implementations of maximum length TLCA. Furthermore, due to the computational intensive nature of the maximum length TLCA searching algorithms, minimal cost machines require the least CPU times and are the only practical machines to find for TLCA of large degree using the computational power available today.

Tables 4.1, 4.2, 4.3, 4.4 and 4.5 list the five types of minimal cost maximum length TLCA machines of degrees up to 60. To be more specific, the second columns consist of the diagonal elements $[d_1 d_2 d_3 \cdots d_n]$ of the transition matrices of the TLCA, and numbers in the first columns represent the corresponding degrees.

To implement a low cost primitive TLCA of degree n of a chosen type (e.g. TLCA(III)), one (a) constructs a binary tree of n cells, and labels the cells from level 0 to level $log_2 n$, and from the left to right at each level, (b) connects the n cells according to the structure definitions given in Figures 3.6, 3.7, 3.8, 3.9 or 3.10 (e.g. using Figure 3.8 for TLCA(III)), and (c) determines the computation rules of the cells using Table 3.2 and Tables 4.1, 4.2, 4.3, 4.4, or 4.5: (c.1) first finds the diagonal elements $[d_1 d_2 \cdots d_n]$ from the respective minimal cost maximum length

TLCA table (e.g. Table 4.3 for TLCA(III)). A d_i in the obtained binary string is corresponding to cell i in the TLCA; then, (c.2) uses Table 3.2. If $d_i = 1$, the computation rule in the parentheses of the chosen TLCA type should be used. Otherwise, i.e. $d_i = 0$, the computation rule outside the parentheses should be taken. Note that cell with even and odd numbers perform different computation rules (see Table 3.2).

Example 4.5 On the fourth row of Table 4.3, the binary string 1100 represents the transition matrix

$$T = \begin{pmatrix} 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix},$$

which defines a minimal cost maximum length TLCA(III) of degree 4. The two 1's of 1100 indicate that the first two cells of the TLCA use the computation rules in parentheses in Table 3.2, i.e. rule 31 for the cell 1 and rule 27 for the cell 2. The last two 0's indicate the last two cells use the computation rules without parentheses in Table 3.2, i.e. rule 15 for the cell 3 and rule 11 for the cell 4. □

For a given type of TLCA of each degree, a brute-force search of maximum-length TLCA is conducted. We first generate all of the TLCA with a single *self-neighboring rule* cell. If no primitive machines are found, we then generate all of the TLCA that have a pair of *self-neighboring* cells. If this fails, we try all of the TLCA which have three cells using *self-neighboring rules* , and so on. For each degree, the search is stopped when the first TLCA with the maximum length cycle is found. The maximum-length TLCA in these tables have an average of four *self-neighboring* cells for the degrees up to 50.

The search is very computational intensive. Here are some examples of CPU time required by Algorithm I running on a SPARC-20:

degree	Type I TLCA
2	10
3	100
4	1010
5	10100
6	110000
7	1000000
8	11100001
9	100000011
10	0100111000
11	00000110000
12	010101100000
13	0000001000000
14	00010000000000
15	011000000000000
16	0000000001000001
17	00010000000000011
18	001100000100000000
19	0001000001000000000
20	00011000001000000000
21	000100000000000000011
22	0011110000000000000000
23	10100100000101000000000
24	101000100000000000000001
25	1010011000001000000000000
26	10000100000001100000000000
27	101001000000010000000000000
28	1100001000000010000000000000
29	10000011000000100000000000000
30	100000100000000000000000000000
31	1000000000000000000000000000000
32	10000001000000010000000000000001
33	100001000000000000000000000000011
34	1000000100000000010000000000000000
35	11000001000000000100000000000000000
36	110000010000000001000000000000000001
37	1100100000010000001000000000000000000
38	01001110000000000000000000000000000000
39	110011100000000000000000000000000000000
41	10000001100000000000000000000000000000011
42	000001100000000000000110000000000000000000
44	00001110010000000000000000000000000000000001
45	100001100000000000000010000000000000000000000
46	1000011000000000000000000000000000000000000000
47	00000110000000000000000000000000000000000000000
48	111000000000000000000000010000000000000000000001
49	1000001000000000000000001110000000000000000000000
50	00100010000000000000000001100000000000000000000000
60	000000010000000000000000000100000000000000000000000000000001

Table 4.1: List of minimal cost maximum-length TLCA(I)

degree	Type II TLCA
2	10
3	111
4	0001
5	10100
6	100001
7	0001100
8	00001000
9	000110000
10	0101000000
11	00100000000
12	100000100000
13	1000001000000
14	01000000000000
15	000110000000000
16	0001000010000000
17	00000000000000011
18	101000010000000000
19	0001100000000000000
20	10100000000000000001
21	000110000010000000000
22	0101100000000000000000
23	00001000000000011000000
24	000110000001100000000000
25	1010001000000000000000011
26	01011000000100000000000001
27	110000100000000000000000000
28	1000011000000000000000000001
29	10000010000000000000000000011
30	101000000000000000000000000001
31	1000000001100110000000000000000
32	10100001100000000000000000000001
33	101000010000000110000000000000000
34	0000100000011001000000000000000000
35	00000001100000000001100000000000000
36	000110000000000000000000000000000001
37	0100100000000000010000000000000000011
38	01001000000000000000000000000000000000
39	010011100000000000000000000000000000000
40	0000100000000000000010000000000000011000
41	00100000000000000011000000000000000000000
42	000100000110000000000000000000000000000001
43	0110100000100000000000000000000000000000000
44	00010000000000000000001000011000000000000000
45	000100000100000000000010000000000000000000000
46	0000000000000110000000000000000000000000000001
47	00011000000000000000000000000000000000000000000
48	010110000000000000000000000000000000000000000001
49	1010000000000000000000000000000000000000000000011
50	10000010000000000000000001100000000000000000000000
60	100000000000000000000000000001000000000000000000000000000000

Table 4.2: List of minimal cost maximum-length TLCA(II)

degree	Type III TLCA
2	10
3	100
4	1100
5	00100
6	000110
7	1000000
8	10010000
9	000010000
10	0000000001
11	10000000000
12	101001000000
13	1000010000000
14	00000110000110
15	011000000000000
16	0010000010000000
17	00100000100000000
18	100110000000000001
19	0100000000000000000
20	00000000010000000000
21	011000000100000000000
22	0000100110000000000001
23	11011000000000000000000
24	111000000000100000000000
25	1110000000001000000000000
26	11100000000000000000000001
27	111001000000000000000000000
28	0100010000001000000000000000
29	11000000000010000000000000000
30	100000111000000000000000000001
31	1000000000000000000000000000000
32	10010001000000001000000000000000
33	100100000000000010000000000000000
34	1000000100011000000000000000000001
35	10000000010000000001100000000000000
36	100100010000000000100000000000000000
37	1001000000000000010000000000000000000
38	11001000100000000000000000000000000001
39	110000000000000000000000000000000000000
40	0010000000000000000010000000000000000001
41	10010000010000000000100000000000000000000
42	100100000100000000000000000000000000000001
43	0010000000000000000000000000000000000000000
44	00110000000000000000010000000000000000000000
45	101000000100000000000010000000000000000000000
46	1000000000111000000000000000000000000000000001
47	00100000000000000000000000000000000000000000000
48	001000000000000000000000100000000000000000000001
49	0000000000110000000000010000000000000000000000000
50	10000000001100000000000000000000000000000000000001
60	000000000000000000000000000010000000000000000000000000000000

Table 4.3: List of minimal cost maximum-length TLCA(III)

degree	Type IV TLCA
2	10
3	111
4	0001
5	01000
6	000001
7	0100000
8	00000001
9	010000010
10	0000100000
11	00010000000
12	101001100000
13	1001000000000
14	10000000000001
15	100100000000000
16	1000100000000001
17	10000000000000000
18	000001100000000001
19	1000000100000000000
20	10100001001000000000
21	000001100100000000000
22	0100010000010000000001
23	00000111000000000000000
24	000000110000000000000001
25	0000001000010000000000000
26	01010000000000000000000001
27	010000000000000000000000000
28	0000000110000000000000000001
29	01000000000001000000000000010
30	010000000000001000000000000001
31	0100000000000000000000000000000
32	00100000100000000000000000000001
33	000000001000000000000000000000000
34	0000000100000000000000000000000100
35	00000000010000000000000000000000000
36	001101000000000000000000000000000001
37	0010000000000000100000000000000000000
38	00010000100000000000000000000000000001
39	000000000100000000000000000000000000000
40	0000000000110000000000000000000000000001
41	00000000001000000000000000000000000000000
42	000000000110000000000000000000000000000100
43	0001000000000000000000000000000000000000000
44	00100100000001000000000000000000000000000001
45	001000000000000000000001000000000000000000000
46	0001000000000001000000000000000000000000000001
47	00100110000000000000000000000000000000000000000
48	000000000000011000000000000000000000000000000001
49	0010010000000000000000000000000000000000000000000
50	00100000000010000000000010000000000000000000000000
60	100101000000001000000000000000000000000000000000000000000001

Table 4.4: List of minimal cost maximum-length TLCA(IV)

degree	Type V TLCA
2	10
3	111
4	0001
5	10100
6	100001
7	0001100
8	00001000
9	000110000
10	0101000000
11	00100000000
12	100000100000
13	1000001000000
14	01000000000000
15	000110000000000
16	0001000010000000
17	00000000000000011
18	101000010000000000
19	0001100000000000000
20	10100000000000000001
21	000110000010000000000
22	0101100000000000000000
23	00001000000000011000000
24	000110000001100000000000
25	1010001000000000000000011
26	01011000000100000000000001
27	110000100000000000000000000
28	1000011000000000000000000001
29	10000010000000000000000000011
30	101000000000000000000000000001
31	1000000001100110000000000000000
32	10100001100000000000000000000001
33	101000010000000110000000000000000
34	0000100000011001000000000000000000
35	00000001100000000011000000000000000
36	000110000000000000000000000000000001
37	0100100000000000010000000000000000011
38	01001000000000000000000000000000000000
39	010011100000000000000000000000000000000
40	0000100000000000000010000000000000011000
41	00100000000000000011000000000000000000000
42	000100000110000000000000000000000000000001
43	0110100000100000000000000000000000000000000
44	00010000000000000000001000011000000000000000
45	000100000100000000000010000000000000000000000
46	0000000000000110000000000000000000000000000001
47	00011000000000000000000000000000000000000000000
48	010110000000000000000000000000000000000000000001
49	1010000000000000000000000000000000000000000000011

Table 4.5: List of minimal cost maximum-length TLCA(V)

1. For a minimal cost maximum-length TLCA of degree 50 having 3 *self-neighboring cells*, the CPU time is about

$$\begin{aligned}(C_{50}^{1}+C_{50}^{2}+C_{50}^{3})\cdot(6.31293232\times 10^{-7})50^{4} &= 503259 \ \textit{seconds}\\ &= 140 \ \textit{hours}\\ &= 5.8 \ \textit{days}.\end{aligned}$$

2. For a minimal cost maximum-length TLCA of degree 60 having 3 *self-neighboring cells*, the CPU time is at least

$$\begin{aligned}(C_{60}^{1}+C_{60}^{2}+C_{60}^{3})\cdot(6.31293232\times 10^{-7})60^{4} &= 866686 \ \textit{seconds}\\ &= 240 \ \textit{hours}\\ &= 10 \ \textit{days}.\end{aligned}$$

3. For a minimal cost maximum-length TLCA of degree 200 having 2 *self-neighboring cells*, the CPU time required is at least

$$\begin{aligned}(C_{200}^{1}+C_{200}^{2})\cdot(6.31293232\times 10^{-7})200^{4} &= 40604780 \ \textit{seconds}\\ &= 11279 \ \textit{hours}\\ &= 470 \ \textit{days}\end{aligned}$$

which is very difficult to finish.

4.5 Maximum Length TLCA vs Primitive Polynomials

Appendix A lists the number of primitive polynomials for degrees up to 24. Table 4.6 lists all maximum length TLCA(III) for degree up to 12, and their corresponding primitive characteristic polynomials. The first column in the table consists of a bit string representing the diagonal elements $[d_1 d_2 d_3 \cdots d_n]$ in the transition matrix of TLCA(III). The second column represents the coefficients (the highest order coefficient first) of the characteristic polynomial of the transition matrix. For example, the TLCA string, 1101, represents the diagonal elements [1101] of its transition matrix of degree 4, and the polynomial string, 11001, represents its characteristic polynomial $x^4 + x^3 + 1$ (see Section 2.2.4).

From the simulation results, it can be seen that

1. The mappings between maximum length TLCA and primitive polynomials are either

$$\begin{array}{ll} 0 - to - 1 & or \\ 2 - to - 1 & or \\ 1 - to - 1\ . & \end{array}$$

2. The number of maximum-length TLCA is much less than the number of primitive polynomials of the same degree when degree $n > 6$.

3. When degree $n > 7$, no primitive characteristic polynomials of the TLCA corresponding to the minimum weight primitive polynomials listed in [5] can be found. Similar results are also found for the other types of TLCA.

The results indicate that unlike LFSRs and LHCA, the majority of TLCA do not have one to one correspondence with primitive polynomials. Thus, there are far fewer choices of machines at each degree. However, in engineering applications, low cost maximum length machines are often the most cost effective ones among all machines. Therefore the lack of choices may not be a problem after all.

degree = 2	
total # of maximal length TLCA	2
total # of primitive polynomials	1
TLCA(III)	characteristic polynomial
01	111
10	111

degree = 3	
total # of maximal length TLCA	2
total # of primitive polynomials	2
TLCA(III)	characteristic polynomial
011	1011
100	1101

degree = 4	
total # of maximal length TLCA	3
total # of primitive polynomials	2
TLCA(III)	characteristic polynomial
1101	11001
0011	10011
1100	10011

degree = 5	
total # of maximal length TLCA	4
total # of primitive polynomials	6
TLCA(III)	characteristic polynomial
00111	111011
00100	101111
11011	100101
11000	110111

degree = 6	
total # of maximal length TLCA	2
total # of primitive polynomials	6
TLCA(III)	characteristic polynomial
000110	1000011
111110	1110011

degree = 7	
total # of maximal length TLCA	6
total # of primitive polynomials	18
TLCA(III)	characteristic polynomial
0111111	10111001
1101100	10000011
0010011	11111101
1010011	10000011
0101100	11111101
1000000	11010011

(continued)

degree = 8	
total # of maximal length TLCA	6
total # of primitive polynomials	16
TLCA(III)	characteristic polynomial
10010001	110000111
01001000	100101011
10010000	100101101
11110110	100101011
11110111	110000111
11101001	111001111

degree = 9	
total # of maximal length TLCA	13
total # of primitive polynomials	48
TLCA(III)	characteristic polynomial
000011111	1101011011
000101111	1101110011
010010000	1010000111
111010000	1001110111
110010011	1100100011
011011111	1110111001
101101111	1111111011
001101111	1010100101
000010000	1101101101
001101100	1000110011
100100000	1011010001
110010000	1111001011
111101111	1001101111

degree = 10	
total # of maximal length TLCA	9
total # of primitive polynomials	60
TLCA(III)	characteristic polynomial
1111100110	11011011111
1110000110	11111110011
1011111110	10100110001
0100000001	10000100111
0001111001	11110001101
0000011001	11010110101
0000000001	11101010101
0001111110	10011010111
1111111110	11101000111

(continued)

degree = 11	
total # of maximal length TLCA	23
total # of primitive polynomials	176
TLCA(III)	characteristic polynomial
01111111111	101100001001
01100111111	100100101001
01001110011	101101000001
01011111111	110010111111
00010110011	110000001011
00001111100	110000001011
10011000000	111000110011
11101001100	101000000111
11001000011	110111010111
11100111111	110100011101
10110001100	110101011001
00100110000	111111101001
01010111100	101101000001
00101110011	101110110111
10101000011	110101011001
10000000000	110100000011
01000000000	110010010111
10100000000	101011011111
00110111100	101110110111
11010001100	110111010111
11011001111	100010011111
11110000011	101000000111
00011000000	101100010001

degree = 12	
total # of maximal length TLCA	9
total # of primitive polynomials	144
TLCA(III)	characteristic polynomial
110011000110	1001110011001
101001000000	1100100011011
111100100110	1101011110101
010010100111	1001100011101
101110111111	1000100110011
110101011000	1001110011001
110111011110	1111110011001
111010111000	1101011110101
010100111001	1001100011101

Table 4.6: Primitive TLCA(III) and their characteristic polynomials

Chapter 5

Pseudorandomness of LFSMs: Theory and Simulation

In Chapter 3, the structures and operations of TLCA are formally defined. This chapter addresses the pseudorandom issues of TLCA. We review theoretical and practical measures of pseudorandom sequences in Section 1. In Section 2, we examine the pseudorandom behavior of TLCA and compare them to existing LFSMs. We present extensive fault simulation results under various fault models using TLCA as the test sequence generator. The final section of this chapter is devoted to special discussions on the pseudorandomness of test sequences and their relation to test coverage.

5.1 Measures of Pseudorandomness

Extensive study on the quality of random and pseudorandom sequences has been reported in the literature. In the first two subsections, we discuss the statistical tests for random numbers, and the more commonly used visual test and correlation test for binary sequences. We then discuss the role of computer simulation in test coverage, and how the pseudorandomness measures are related to the quality of the testing of digital circuits.

5.1.1 Equidistribution Test

The equidistribution test is one of the statistical tests for a sequence of random integer and real numbers [24, pp. 54-65]. It tests if numbers in a sequence are uniformly distributed among categories $0, 1, \cdots, k-1$. The test has been adapted to measure the pseudorandom quality of binary sequences [22, 23]. In these applications, a binary sequence is treated as a sequence of integers. Then, the chi-square test (χ^2) is used to compare the observed and expected distributions, and to provide a pass/fail result.

We conduct a set of equidistribution tests on binary sequences generated by three types of LFSMs: LFSR, LHCA and TLCA. To examine the impact of different initial states of LFSMs and different number of category values, k, on the final statistical results, three different initial states and four randomly generated k values are applied to two machines of different degrees for each type. The detailed test procedures are as follows:

1. generate N successive n-bit binary patterns using a maximum length LFSR, LHCA or TLCA;
2. convert the N binary patterns into their corresponding decimal values, where the left most bits of the binary patterns are the most significant bits;
3. reduce the decimal values to the numbers ranging from 0 to $k-1$ by taking modulo k operations, where k is an arbitrarily chosen prime number. The decimal numbers in the k categories are called the *observed results*. Let Y_i be the total number of observed results that fall into category i, that is, the decimal values that equal to i;
4. compute the expected values Np_i, assuming that the distribution of the test is the probability $p_i = \frac{1}{k}$ for each category i;
5. apply the chi-square test (χ^2) for the comparison

$$V = \sum_{i=0}^{k-1} \frac{(Y_i - Np_i)^2}{(Np_i)} ; \qquad (5.1)$$

6. repeat steps 1 to 5 three times for three difference initial binary states;

7. repeat steps 1 to 6 four times for four different values of k.

Once a value V is obtained, one can refer to the chi-square distribution table in [24, pp. 39] to find the closest corresponding probability P. That is, the table entry x in row v and column P should be closest value to V. The v is the degree of freedom and equals to $k-1$. The interpretation is that the quantity V will be greater than x with probability P. For example, the 5 percent entry in row 10 is 18.31; this says we will have $V > 18.31$ only 5% of the time.

The criteria to fail and pass the randomness test are suggested in [24, pp. 39-40]:

1. if the value of V lies beyond columns 1% to 99%, then the sequence is rejected as a random sequence;

2. if the value of V lies between the columns 1 and 5% or 95 and 99%, then the sequence is called "suspect" in randomness behavior;

3. if V lies between columns 10 and 90%, then the sequence is said to be fully acceptable as a random sequence.

Tables 5.1, 5.2, and 5.3 list the results of the equidistribution test for the three types of LFSMs of degrees 32 and 60. The binary representations of the LFSMs configurations can be found in Appendix E. The length of the sequence under test is 10,000, i.e. $N = 10,000$. It can be seen that all the LFSRs fail the tests, and both LHCA and TLCA pass the tests. The statistical tests show that 1-d and 2-d LCA are better than LFSRs in pseudorandom behavior, which is consistent with the previous work reported in [23]. However, the tests do not provide quantitative measures of pseudorandomness within LHCA and TLCA types, and between the two types. In Section 5.2, we will show that fault simulation is a more reliable measure of pseudorandomness of binary sequences in testing applications.

5.1.2 Visual Test and Correlation Test

Two binary sequence test approaches used by some researchers and test engineers are known as the *visual test* and the *correlation test.*

The **visual test** employs graphical outputs of a binary sequence to observe the regularities and randomness. A binary sequence of t n-bit binary patterns can be treated as a graph with $n \times t$ tiles, where n is the length of the binary patterns and t is the time space. If one uses a star and a blank to represent the two types of tiles to build the graph, and uses the star to denote a logic 1 output and the blank for a logic 0, a graphical output of a binary sequence can be produced.

Figure 5.1 shows the visual test graphs generated by the three maximum length LFMS: LFSR, LHCA and TLCA(III) of degree 20. The characteristic polynomial of the LFSR is $x^{20}+x^{3}+1$. The LHCA is from [40]. The TLCA(III) is from Table 4.3. All the three machines produce one 20-bit wide binary pattern at a time, and run for 80 consecutive clock cycles, starting from the same initial pattern 00000000010100000000. Figure 5.2 displays the binary patterns of 240 consecutive clock cycles in three blocks generated by the same LFSR. The regularity of the LFSR patterns can be clearly identified in Figures 5.1 (a) and 5.2. In comparison, in Figure 5.1 (b), the patterns of the LHCA graph look more random. However, there are some regular triangular shaped patterns scattered across the picture. Figure 5.1 (c) depicts the 80 patterns generated by a 20-bit TLCA(III), where almost no regularities can be visually detected.

The **correlation Test** reveals the dependence of one pattern in a binary sequence over the others. There are two correlation measures: auto and cross correlation. Let $< X >= X_0, X_1, \ldots, X_{n-1}$ and $< Y >= Y_0, Y_1, \ldots, Y_{n-1}$ represent two patterns of length of n bits. The cross correlation coefficient, denoted by

Machine of degree	# of Categories	χ^2 V	Probability P (%)	Random?
initial state		00· · ·001		
32	11	6.94	70	Y
	37	22.71	85	Y
	61	56.99	60	Y
	89	72.47	70	Y
initial state		100· · ·001		
32	11	2.67	98	S
	37	23.51	85	Y
	61	49.07	75	Y
	89	104.38	20	Y
initial state		111111111100· · ·001		
32	11	22.50	2	S
	37	9×10^{15}	0	N
	61	54.07	60	Y
	89	98.87	28	Y
initial state		00· · ·001		
60	11	149.46	0	N
	37	114.56	0	N
	61	121.51	0	N
	89	107.32	20	Y
initial state		100· · ·001		
60	11	153.45	0	N
	37	114.87	0	N
	61	124.50	0	N
	89	109.83	20	Y
initial state		111111111100· · ·001		
60	11	6.33	80	Y
	37	20.82	95	S
	61	98.23	4	S
	89	543.17	0	N

Table 5.1: Equidistribution test for the LFSRs (Y=Yes, S=Suspect, N=No)

Machine of degree	# of Categories	χ^2 V	Probability P (%)	Random?
initial state		00···001		
32	11	6.58	80	Y
	37	33.84	60	Y
	61	79.34	15	Y
	89	101.82	25	Y
initial state		100···001		
32	11	8.63	55	Y
	37	43.06	30	Y
	61	45.42	80	Y
	89	89.93	4	S
initial state		111111111100···001		
32	11	13.84	20	Y
	37	37.76	45	Y
	61	71.81	25	Y
	89	60.69	96	S
initial state		00···001		
60	11	8.09	65	Y
	37	45.41	25	Y
	61	62.29	45	Y
	89	97.67	30	Y
initial state		100···001		
60	11	7.63	70	Y
	37	25.34	85	Y
	61	59.35	50	Y
	89	84.39	60	Y
initial state		111111111100···001		
60	11	6.78	75	Y
	37	19.56	96	S
	61	52.02	70	Y
	89	86.99	60	Y

Table 5.2: Equidistribution test for the LHCA (Y=Yes, S=Suspect, N=No)

Machine of degree	# of Categories	χ^2 V	Probability P (%)	Random?
initial state		00···001		
32	11	15.23	10	Y
	37	34.87	60	Y
	61	62.51	45	Y
	89	100.13	25	Y
initial state		100···001		
32	11	4.15	92	Y
	37	35.54	55	Y
	61	55.77	70	Y
	89	81.49	10	Y
initial state		111111111100···001		
32	11	7.08	70	Y
	37	63.90	3	S
	61	55.34	70	Y
	89	99.85	25	Y
initial state		00···001		
60	11	7.99	65	Y
	37	47.28	20	Y
	61	74.24	20	Y
	89	89.89	50	Y
initial state		100···001		
60	11	4.54	90	Y
	37	37.17	52	Y
	61	63.16	45	Y
	89	110.33	20	Y
initial state		111111111100···001		
60	11	10.92	30	Y
	37	34.50	60	Y
	61	53.97	68	Y
	89	90.16	45	Y

Table 5.3: Equidistribution test for the TLCA(III) (Y=Yes, S=Suspect, N=No)

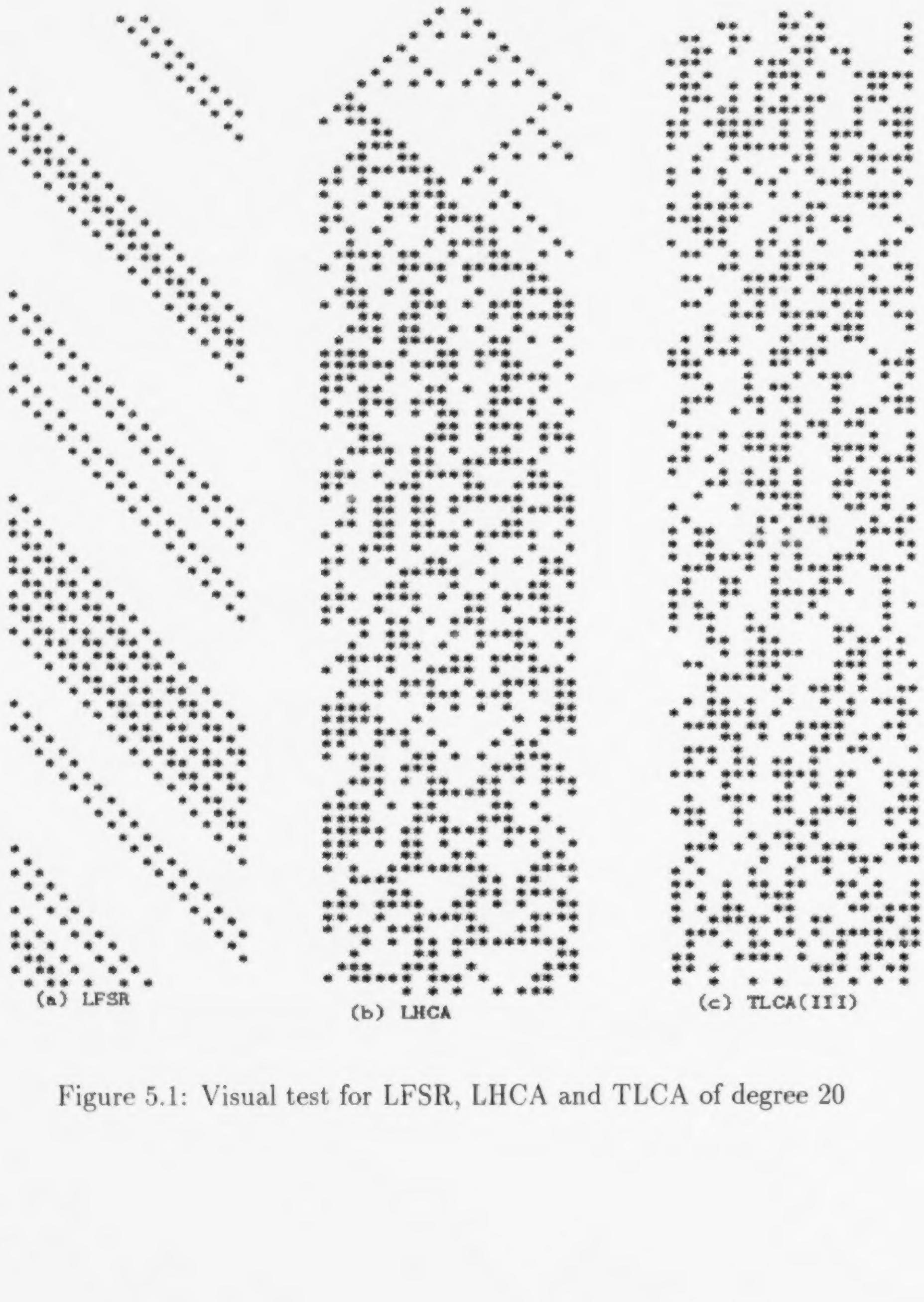

(a) LFSR (b) LHCA (c) TLCA(III)

Figure 5.1: Visual test for LFSR, LHCA and TLCA of degree 20

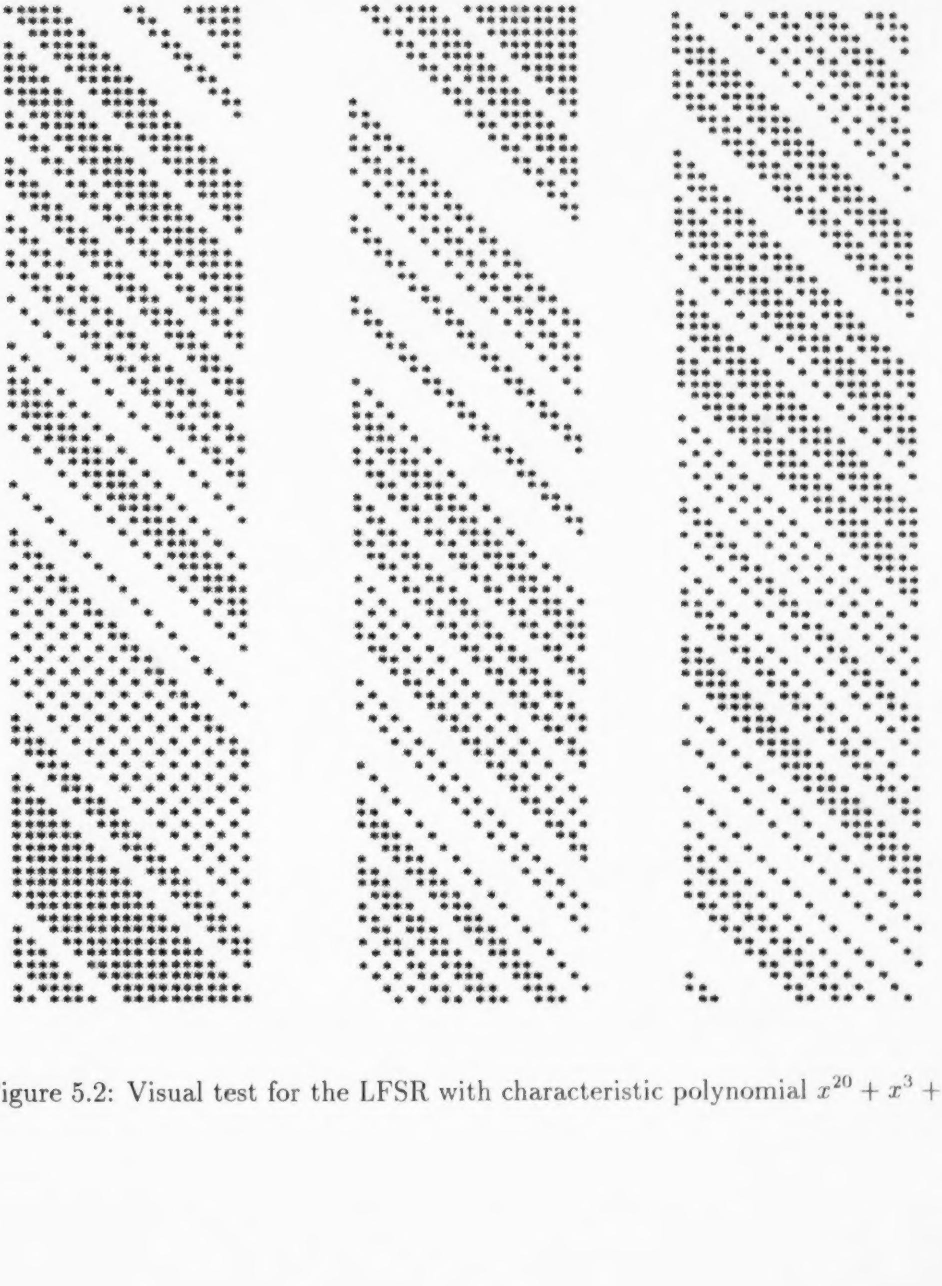


Figure 5.2: Visual test for the LFSR with characteristic polynomial $x^{20} + x^3 + 1$

CO_c, measures the dependencies between $< X >$ and $< Y >$ and is calculated by

$$CO_c = \frac{n \sum_{i=0}^{n-1} (X_i Y_i) - (\sum_{i=0}^{n-1} X_i \sum_{i=0}^{n-1} Y_i)}{\sqrt{(n \sum_{i=0}^{n-1} X_i^2 - (\sum_{i=0}^{n-1} X_i)^2)(n \sum_{i=0}^{n-1} Y_i^2 - (\sum_{i=0}^{n-1} Y_i)^2)}} \tag{5.2}$$

The auto correlation coefficient, denoted by CO_a, measures the correlation of elements within a pattern. For a pattern $< X >$, CO_a is computed using the same formula as that of CO_c except that the sequence $< Y >$ is replaced by the pattern $< X >$ cyclically shifted i positions, where i ranges from $0, \ldots, n-1$.

The coefficients CO_a and CO_c lie in the range -1 to +1. A coefficient of near zero indicates that the patterns are independent of each other or are not correlated. On the other hand, a value of close to ±1 tells us that the patterns are dependent or are correlated.

Only the absolute value of the correlation coefficients, ranged from 0 to +1, needs to be considered for the computation of the correlation coefficients of binary patterns. Under this situation, the equation 5.2 can be simplified to

$$CO_c = \frac{n \sum_{i=0}^{n-1} (X_i Y_i) - (\sum_{i=0}^{n-1} X_i \sum_{i=0}^{n-1} Y_i)}{\sqrt{(n \sum_{i=0}^{n-1} X_i - (\sum_{i=0}^{n-1} X_i)^2)(n \sum_{i=0}^{n-1} Y_i - (\sum_{i=0}^{n-1} Y_i)^2)}}. \tag{5.3}$$

The following procedures are used to compute the correlation coefficients of n-bit binary patterns generated by one of a maximum length LFSR, 1-d LHCA and TLCA.

1. Successively generate n n-bit binary patterns.
2. Test the first pattern generated at time $t = 0$ (denoted by Q_0) against all the patterns Q_t including itself by using the following method:

```
FOR $t$ FROM 0 TO $n - 1$ DO
    FOR $j$ FROM 0 TO $n - 1$ DO
$CO_c[t,j] = \frac{n \sum_{i=0}^{n-1} (Q_0[i] Q_t[i+j \mod n]) - (\sum_{i=0}^{n-1} Q_0[i] \sum_{i=0}^{n-1} Q_t[i])}{\sqrt{(n \sum_{i=0}^{n-1} Q_0[i] - (\sum_{i=0}^{n-1} Q_0[i])^2)(n \sum_{i=0}^{n-1} Q_t[i+j \bmod n] - (\sum_{i=0}^{n-1} Q_t[i+j \bmod n])^2)}}$
    END
END
```

Note that CO_c is a two dimensional array of size$(0..n-1, 0..n-1)$. When $t = 0$, the method computes the auto correlations of the sequence Q_0.

Figures 5.3, 5.4, and 5.5 demonstrate the auto and cross correlation of maximum length LFSR, LHCA and TLCA(III) of degree 36 in 3-dimensional plots, respectively. The X axis (labeled *time*) represents the time displacement since different patterns are generated at different times. The Y axis (labeled *sequence*) represents sequence displacement, that is, how many bits shifted for the same pattern. The correlation coefficient is shown in the Z axis. The auto correlation can be read from the curve on the plane vertical to the *time* axis. The cross correlation of the pattern cyclically shifted some bits can be read from the values on the plane vertical to *sequence* axis.

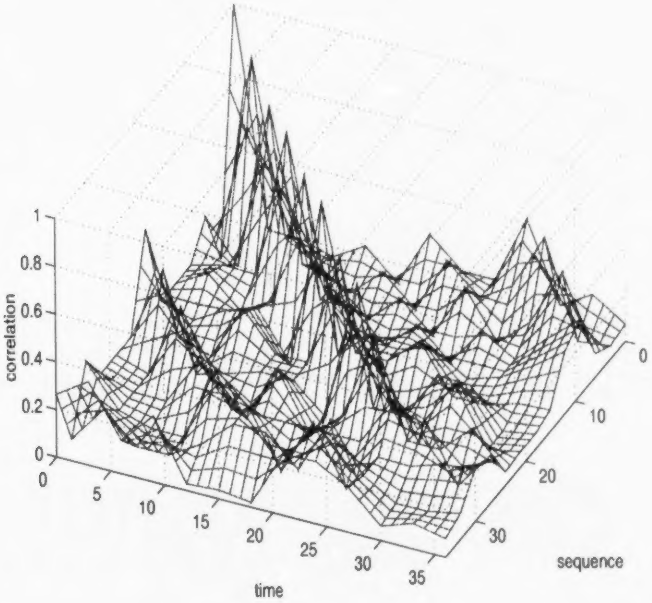


Figure 5.3: Auto and cross correlation test for LFSR of degree 36

The high correlation for patterns from the LFSRs is indicated by the diagonal ridge in Figure 5.3. This is because the pattern Q_0 and the pattern Q_t shifted t positions are almost identical for most t's. The regularity of the patterns generated can also be seen by the wave-like patterns of the correlation coefficients

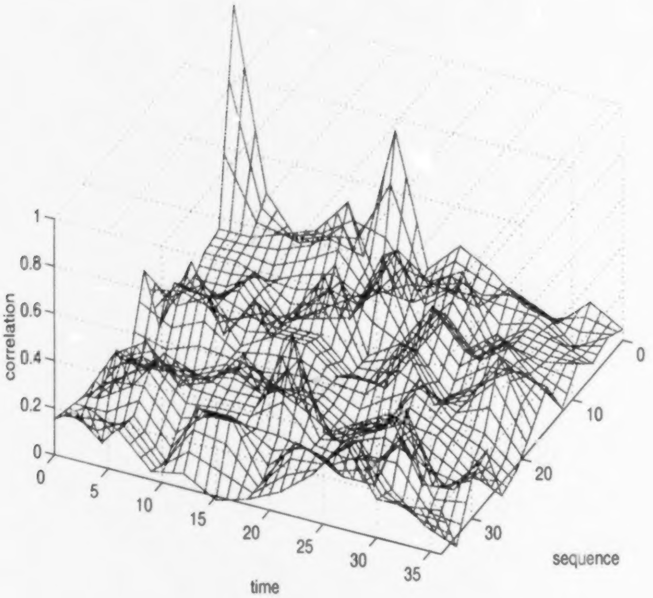


Figure 5.4: Auto and cross correlation test for LHCA of degree 36

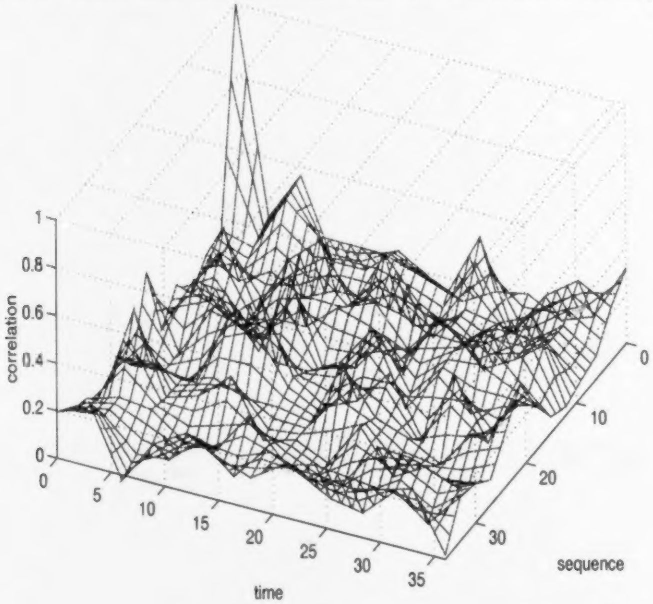


Figure 5.5: Auto and cross correlation for TLCA(III) of degree 36

on the two sides of the main diagonal ridge. Figures 5.4 and 5.5 show the correlation coefficients for the patterns generated from 1-d LHCA and TLCA(III) respectively. They both show no diagonal ridge and wave-like regularity of the correlation coefficients.

5.1.3 The Role of Fault Simulation

Statistical tests were originally designed for randomness tests of integer or real numbers. It has been a long, unresolved, argument in the testing community on the value of these tests for binary sequences.

Extensive visual tests have been conducted to examine the pseudorandom properties of binary sequence generators. It has been reported that the pseudorandom sequences generated by RLCA (2-d machines) are superior to that of LHCA, and in turn that of LHCA are better than that of LFSRs [22, 23]. However, visual tests, similar to the equidistribution statistical tests discussed in Section 5.1.1, do not provide quantitative measures of the pseudorandomness within each type of the machines and among the three different types. Correlation tests suffer a similar problem. Visually, one can claim that a binary sequence produced by one machine has a higher correlation than another. Again, there is no quantitative measure which can be used to differentiate the degree of correlation in binary sequences.

The quality of test sequences greatly influences the quality of the testing of digital circuits. In practice, fault simulation is commonly used to determine the quality of a test set. Fault simulation consists of simulating a circuit in the presence of faults. Physical defects of digital circuitry are represented by a number of fault models. The commonly used fault models have been discussed in Chapter 2 of this thesis. Comparing the fault simulation results with those of the fault-free simulation of the same circuit simulated with the same applied test set T, one can determine the faults detected by T. Therefore, the quality of a test sequence can also be measured by the fault coverage of test T. A common practice is to apply a test sequence to a set of standard benchmark circuits, then conduct the fault simulation for a chosen fault model and evaluate the fault coverage.

It has been reported that the quality measures of test sequences obtained from fault simulation often contradict the theoretical quality measures introduced in the previous two subsections. One intuitive explanation is that the tests perform independent measures of a sequence, while fault simulation examines the consequences of a test application. In other words, something is missing in the measures performed by the theoretical tests. A notable attempt made to relate a test sequence to the test coverage of digital circuits is the two-pattern transition property [17, 39]. It is proposed for delay fault coverage. The term *transition coverage* is introduced to indicate the number of possible transitions in the consecutive pairs of test patterns. It is claimed that the higher the transition coverage, the better the fault coverage. However, we will show later in Section 5.3 that our simulation results, using TLCA as test generators, provide counter examples of this claim.

It appears that fault simulation is the only means that provides an unarguable measure of the quality of a test sequence. In the next section, we will present our fault simulation results on the quality and effectiveness of test sequences produced by various LFSMs on the standard ISCAS '85 benchmark circuits.

5.2 Fault Simulation

In this section, we first introduce the the simulation environment. Then we present the simulation results on test coverage for stuck-at, transition, and stuck-open fault models separately. Discussions and observations on the simulation results are also given.

5.2.1 Simulation Environment

5.2.1.1 Fault Simulator

A fault simulator, called *sim3*, is used in this investigation. *Sim3* was developed in the VLSI Research Center, University of Victoria, Canada. It has been tested and used by a number of researchers. The results of their simulations have been reported in technical journals and international conferences [38, 41].

Sim3 employs the fault collapsing technique to group equivalent faults in the original fault set in order to form a reduced fault set [1, pp. 216]. Fault simulation then is performed for all faults in the reduced set, and the simulation time is reduced. Moreover, *sim3* uses the Parallel Pattern Single Fault Propagation (PPSFP) technique [34] in the implementation. It allows several binary test vectors to be applied simultaneously, hence, speeding up the simulation process. The main features of *sim*3 are:

1. it supports three fault models: stuck-at, transition, and stuck-open fault models;
2. it provides several choices of built-in binary test pattern generators, including LFSR(I), LFSR(II) and LHCA;
3. it is capable of accepting test patterns supplied externally by an user.

The manual page of *sim*3 can be found in Appendix C.

5.2.1.2 ISCAS '85 Benchmark Circuits

ISCAS '85 benchmarks consist of ten combinational circuits and were issued by the 1985 International Symposium on Circuits and Systems [16]. They have been widely used by researchers and engineers to measure and compare the effectiveness of test generation algorithms and test sequences. A brief description of the characteristics of the circuits can be found in Table 5.4.

Circuit Name	Circuit Function	Total Gates	Input Lines	Output Lines
C432	Priority Decoder	160 (18 EXOR)	36	7
C499	ECAT	202 (104 EXOR)	41	32
C880	ALU and Control	383	60	26
C1355	ECAT	546	41	32
C1908	ECAT	880	33	25
C2670	ALU and Control	1193	233	140
C3540	ALU and Control	1669	50	22
C5315	ALU and Selector	2307	178	123
C6288	16-bit Multiplier	2406	32	32
C7552	ALU and Control	3512	207	108

Table 5.4: Characteristics of ISCAS '85 Benchmark Circuits

5.2.2 Simulation Results

As stated earlier, fault simulation can be used to determine the fault coverage of a test sequence. Extensive fault simulation is conducted under three fault models: stuck-at, transition and stuck-open.

We choose to use 102,000 test vectors in this experiment. This number is taken from [35] as a typical number used in testing applications. We use primitive LFSRs, LHCA, and TLCA as the test pattern generators. It is assumed that the length of a sequence generator equals the number of inputs of a benchmark circuit under evaluation. In the tables below, we give the types of the generators only, such as LFSR, LHCA, etc. The intention is to examine the different types of machines. However, there are many different choices of machines under each type. For example, there are 2,048 primitive polynomials in degree 16, and each can be used as the generator. Taking the hardware cost of the implementation of machines into consideration, we choose to use low cost machines only. Similarly, for low cost machines of a given degree, there may exist more than one choice. The actual LFSMs of each type used in the simulations are listed in Appendix E.

For all the simulation results except those in Section 5.2.2.4, an initial state of $00\cdots001$ is used for the test pattern generators. In Section 5.2.2.4, we explore the effect of initial states of test generators on test coverage. The simulation results show that the impact of chosen initial states on the test coverage is negligible for the test lengths used in practical today ($> 100,000$).

Given a benchmark circuit, the following simulation process is applied to all the simulations: *sim3* enumerates each fault under a given fault model, and injects one fault to the fault-free circuit at a time. Then, *sim3* applies the test patterns generated by a chosen LFSM (for example, an LFSR) to the faulty circuit, and evaluates the output responses against the fault-free outputs. If an output responding to a test vector is equal to the fault-free output of the same test input, the injected fault is said to be *undetectable* by the test vector. If the fault can not be detected by any of the 102,000 test vectors generated, the fault is undetectable by the given test sequence; otherwise, it is *detectable*.

The fault coverage of a benchmark circuit under a chosen fault model is defined as the ratio of the number of detected faults over the total number of simulated faults (N), where the number of detected faults is equal to the total number of simulated faults minus the number of undetected faults (U), i.e.

$$Fault\ coverage \quad = \quad \frac{N-U}{N} \times 100\%.$$

Note that a 100% fault coverage may still fail to detect faults outside a considered fault model.

5.2.2.1 Single Stuck-at Fault Coverage

Table 5.5 lists the single stuck-at fault coverage of the fixed test length (102,000) generated by the LFSR(II), LHCA, and TLCA. The results show that the three LFSMs share the same single stuck-at fault coverage. The results obtained for LFSR(II) and LHCA is consistent with the investigation reported in [10, 23].

It is well known that fault coverage has close correlation with test length.

$C$432: Priority Decoder, 36 inputs and 7 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	560	40	92.86
1-d LHCA	560	40	92.86
TLCA(I)	560	40	92.86
TLCA(II)	560	40	92.86
TLCA(III)	560	40	92.86
TLCA(IV)	560	40	92.86
TLCA(V)	560	40	92.86

$C$499: ECAT, 41 inputs and 32 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1158	8	99.31
1-d LHCA	1158	8	99.31
TLCA(I)	1158	8	99.31
TLCA(II)	1158	8	99.31
TLCA(III)	1158	8	99.31
TLCA(IV)	1158	8	99.31
TLCA(V)	1158	8	99.31

$C$880: ALU and Control, 60 inputs and 26 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	942	0	100.00
1-d LHCA	942	0	100.00
TLCA(I)	942	0	100.00
TLCA(II)	942	0	100.00
TLCA(III)	942	0	100.00
TLCA(IV)	942	0	100.00
TLCA(V)	942	0	100.00

$C$1355: ECAT, 41 inputs and 32 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1574	8	99.49
1-d LHCA	1574	8	99.49
TLCA(I)	1574	8	99.49
TLCA(II)	1574	8	99.49
TLCA(III)	1574	8	99.49
TLCA(IV)	1574	8	99.49
TLCA(V)	1574	8	99.49

(continued)

$C1908$: ECAT, 33 inputs and 25 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1879	9	99.52
1-d LHCA	1879	9	99.52
TLCA(I)	1879	9	99.52
TLCA(II)	1879	9	99.52
TLCA(III)	1879	9	99.52
TLCA(IV)	1879	9	99.52
TLCA(V)	1879	9	99.52

$C3540$: ALU and Control, 50 inputs and 22 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	3428	137	96.00
1-d LHCA	3428	137	96.00
TLCA(I)	3428	137	96.00
TLCA(II)	3428	137	96.00
TLCA(III)	3428	137	96.00
TLCA(IV)	3428	137	96.00
TLCA(V)	3428	137	96.00

$C6288$: 16-bit Multiplier, 32 inputs and 32 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	7744	34	99.56
1-d LHCA	7744	34	99.56
TLCA(I)	7744	34	99.56
TLCA(II)	7744	34	99.56
TLCA(III)	7744	34	99.56
TLCA(IV)	7744	34	99.56
TLCA(V)	7744	34	99.56

Table 5.5: Stuck-at fault simulation results for LFSR, LHCA and TLCA

In general, the longer the test length, the higher the fault coverage. To further investigate the effectiveness of the test sequences generated by different LFSM structures, fault simulations with different test lengths are conducted. Table 5.6 shows the stuck-at fault coverage as a function of test length using different pseudorandom sources, for the benchmark circuits C1355, C3540, and C880. It can be seen that at short test lengths, TLCA perform better than LHCA, and both are better than LFSR. However, if test lengths are sufficiently long, the fault coverage of the machines of all types is asymptotic to a constant. This implies that the effectiveness of the machines as a sequence generator is best judged at shorter lengths.

5.2.2.2 Transition Fault Coverage

Table 5.7 lists the transition fault coverage of the fixed length test patterns generated from the LFSR(II), 1-d CA and TLCA. Comparing TLCA with 1-d LHCA, it can be seen that the former has the same, better, and slightly worse performance than the latter at a length of 32, 33 and 36, respectively. For longer machines (≥ 41), TLCA are always better than LHCA. Comparison is also made between TLCA and rectangular-structured LCA. Only two case studies (for lengths 32 and 36 only) were reported in [23]. The fault coverage of TLCA is comparable to that of RLCA at degree 32, but is a little worse than RLCA at degree 36. Overall, all 1-d and 2-d CA perform much better than LFSRs.

The performance of each type of test sequence generator is also examined against test lengths. Figures 5.6, 5.7 and 5.8 illustrate the transition fault coverage as a function of the number of test patterns, using the benchmark circuits C1355, C3540, and C880. The three circuits are the largest we can simulate in this set of benchmarks due to the limitation of computer power. The solid, dash-dotted, and dotted lines show the fault coverage of TLCA(III), 1-d LHCA, and LFSRs, respectively. The '+' points represent the performances of TLCA(I) and the 'o' represents TLCA(II). The tabular data used to generate the plots can be found

$C1355$: ECAT, 41 inputs and 32 outputs

Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	77.32	76.24	88.25	89.39	87.99
500	89.64	92.38	95.43	96.12	94.79
700	94.60	93.20	96.76	97.78	96.32
1000	98.03	95.81	98.28	99.30	97.65
2000	99.36	99.30	99.49	99.40	99.49
4000	99.49	99.49	99.49	99.49	99.49
5000	99.49	99.49	99.49	99.49	99.49
7000	99.49	99.49	99.49	99.49	99.49
102000	99.49	99.49	99.49	99.49	99.49

$C3540$: ALU and Control, 50 inputs and 22 outputs

Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	51.40	76.72	81.53	82.44	79.64
500	90.34	90.43	91.45	92.01	91.48
1000	94.71	94.11	94.22	94.75	94.19
2000	95.60	95.54	95.33	95.60	95.48
4000	95.71	95.83	95.71	95.80	95.83
5000	95.80	95.89	95.83	95.85	95.92
7000	95.89	95.95	95.89	95.86	95.92
10000	95.92	95.97	95.89	95.89	95.95
20000	95.97	95.97	95.97	95.92	95.97
60000	96.00	96.00	96.00	96.00	96.00
102000	96.00	96.00	96.00	96.00	96.00

$C880$: ALU and Control, 60 inputs and 26 outputs

Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
200	45.75	91.19	94.90	94.06	93.52
400	53.29	95.44	96.60	96.50	95.97
500	58.39	96.60	96.71	96.82	96.39
700	63.91	97.24	97.24	96.82	96.92
1000	77.60	97.77	97.98	97.35	97.45
2000	88.96	98.62	99.26	98.94	99.04
5000	96.18	99.04	99.56	99.58	99.58
10000	99.47	99.58	99.89	99.58	99.89
20000	99.79	99.88	99.90	99.79	99.89
60000	100.0	100.0	100.0	100.0	100.0
102000	100.0	100.0	100.0	100.0	100.0

Table 5.6: Stuck-at fault coverages at different lengths

$C432$: Priority Decoder, 36 inputs and 7 outputs

	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	772	97	87.44
1-d LHCA	772	44	94.30
TLCA(I)	772	48	93.78
TLCA(II)	772	48	93.78
TLCA(III)	772	63	91.84
TLCA(IV)	772	67	91.32
TLCA(V)	772	44	94.30

	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1572	41	97.39
1-d LHCA	1572	9	99.43
TLCA(I)	1572	8	99.49
TLCA(II)	1572	9	99.43
TLCA(III)	1572	8	99.49
TLCA(IV)	1572	11	99.30
TLCA(V)	1572	10	99.36

$C880$: ALU and Control, 60 inputs and 26 outputs

	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1313	48	96.34
1-d LHCA	1313	7	99.47
TLCA(I)	1313	9	99.31
TLCA(II)	1313	5	99.62
TLCA(III)	1313	3	99.77
TLCA(IV)	1313	6	99.54

$C1355$: ECAT, 41 inputs and 32 outputs

	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	2092	79	96.22
1-d LHCA	2092	35	98.33
TLCA(I)	2092	34	98.37
TLCA(II)	2092	33	98.42
TLCA(III)	2092	31	98.56
TLCA(IV)	2092	34	98.37
TLCA(V)	2092	40	98.09

(continued)

$C1908$: ECAT, 33 inputs and 25 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	2497	65	97.40
1-d LHCA	2497	42	98.32
TLCA(I)	2497	28	98.88
TLCA(II)	2497	39	98.44
TLCA(III)	2497	19	99.24
TLCA(IV)	2497	48	98.08
TLCA(V)	2497	49	98.04

$C3540$: ALU and Control, 50 inputs and 22 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	4707	484	89.72
1-d LHCA	4707	307	93.48
TLCA(I)	4707	308	93.46
TLCA(II)	4707	222	95.28
TLCA(III)	4707	233	95.05
TLCA(IV)	4707	271	94.24

$C6288$: 16-bit Multiplier, 32 inputs and 32 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	10128	102	98.99
1-d LHCA	10128	69	99.32
TLCA(I)	10128	77	99.24
TLCA(II)	10128	77	99.24
TLCA(III)	10128	70	99.31
TLCA(IV)	10128	68	99.33
TLCA(V)	10128	77	99.24

Table 5.7: Transition fault simulation results for LFSR, LHCA and TLCA

in Appendix D.

It appears that for the transition fault model TLCA(III) always have superior performance than LHCA, regardless the test lengths. TLCA(I) gives a comparable fault coverage as LHCA. TLCA(II) are never inferior to LHCA. Both LHCA and TLCA(III) are better than LFSRs.

5.2.2.3 Stuck-open Fault Coverage

Table 5.8 lists the stuck-open fault coverage of the fixed length test patterns generated from the LFSR(II), the 1-d CA and the TLCA. It is observed that TLCA are superior to LHCA except at the length 36.

Comparing TLCA with the rectangular-structured LCA (only two cases were reported in [23]), the stuck-open fault coverage of TLCA is comparable to that of RLCA at degree 32, but is slightly worse at degree 36. Again, all 1-d and 2-d CA perform much better than LFSRs.

Similarly, Figures 5.9, 5.10 and 5.11 depict the stuck-open fault coverage as a function of the number of test patterns. The tabular data used to generate the plots are given in Appendix D. It is shown that for the stuck-open model, TLCA(I), TLCA(II), and TLCA(III) perform better than LHCA, independent of the test lengths. The LHCA and the three types of TLCA are superior to the LFSR.

5.2.2.4 Fault Coverage vs Initial States of Test Pattern Generators

There are two scenarios to be considered: the effect of initial states of test pattern generators on test coverage of digital circuits when test lengths are shorter and longer than (or equal to) the period of the test pattern generators. The latter refers to the simulations where exhaustive input patterns are applied to a CUT. Therefore, the test coverage is completely independent of the initial states of the generators. However, in test applications, it is often impractical to have exhaustive testing, even for circuits of moderate sizes (e.g. with 100 inputs) due

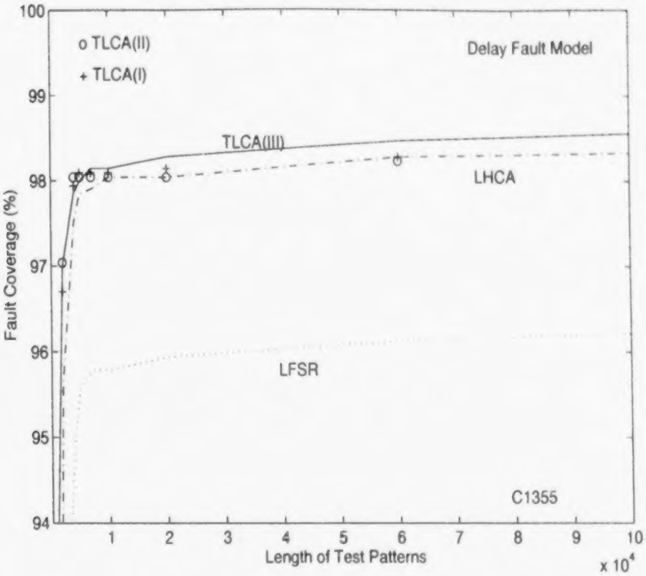


Figure 5.6: Transition fault coverage vs test length using C1355 (41 inputs)

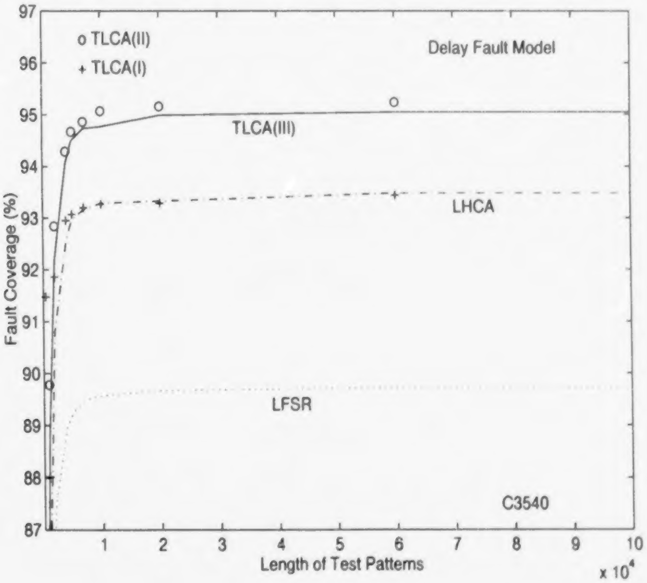


Figure 5.7: Transition fault coverage vs test length using C3540 (50 inputs)

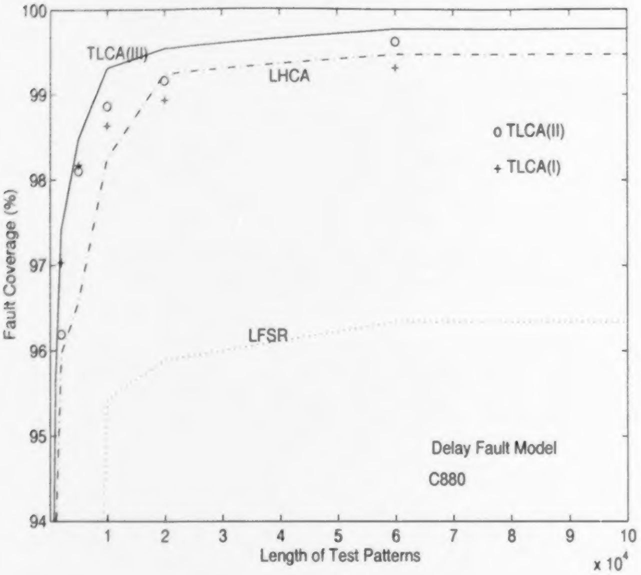


Figure 5.8: Transition fault coverage vs test length using C880 (60 inputs)

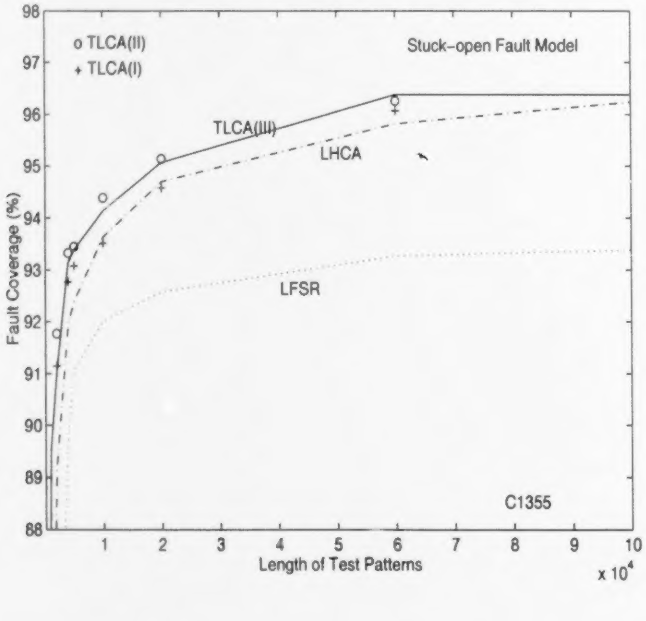


Figure 5.9: Stuck-open fault coverage vs test length using C1355 (41 inputs)

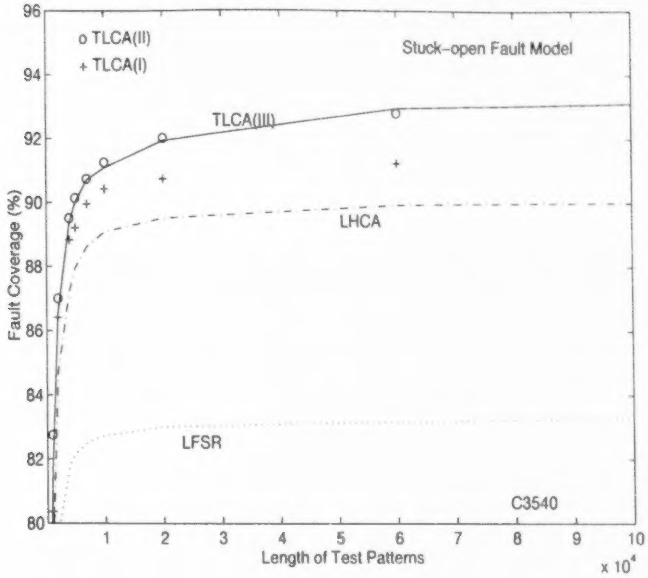


Figure 5.10: Stuck-open fault coverage vs test length using C3540 (50 inputs)

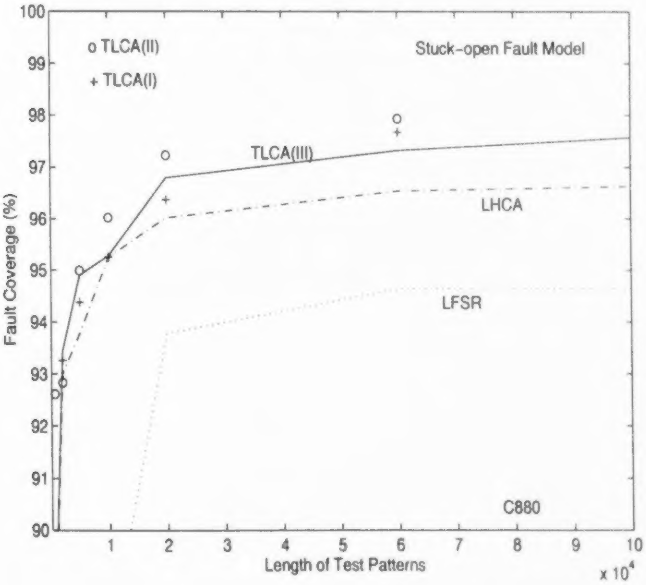


Figure 5.11: Stuck-open fault coverage vs test length using C880 (60 inputs)

$C$432: Priority Decoder, 36 inputs and 7 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	670	81	87.91
1-d LHCA	670	44	93.43
TLCA(I)	670	53	92.09
TLCA(II)	670	53	92.09
TLCA(III)	670	66	90.15
TLCA(IV)	670	60	91.04
TLCA(V)	670	52	92.24

$C$499: ECAT 41 inputs and 32 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1708	84	95.08
1-d LHCA	1708	34	98.01
TLCA(I)	1708	30	98.24
TLCA(II)	1708	32	98.13
TLCA(III)	1708	35	97.95
TLCA(IV)	1708	37	97.83
TLCA(V)	1708	34	98.01

$C$880: ALU and Control, 60 inputs and 26 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1157	62	94.64
1-d LHCA	1157	39	96.63
TLCA(I)	1157	25	97.84
TLCA(II)	1157	23	98.01
TLCA(III)	1157	28	97.58
TLCA(IV)	1157	41	96.46

$C$1355: ECAT, 41 inputs and 32 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	1604	106	93.39
1-d LHCA	1604	60	96.26
TLCA(I)	1604	56	96.51
TLCA(II)	1604	56	96.51
TLCA(III)	1604	58	96.38
TLCA(IV)	1604	59	96.32
TLCA(V)	1604	63	96.07

(continued)

*C*1908: ECAT, 33 inputs and 25 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	2094	260	87.58
1-d LHCA	2094	237	88.68
TLCA(I)	2094	218	89.59
TLCA(II)	2094	228	89.11
TLCA(III)	2094	139	93.36
TLCA(IV)	2094	224	89.30
TLCA(V)	2094	196	90.64

*C*3540: ALU and Control, 50 inputs and 22 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	4708	788	83.26
1-d LHCA	4708	470	90.02
TLCA(I)	4708	406	91.38
TLCA(II)	4708	331	92.97
TLCA(III)	4708	324	93.12
TLCA(IV)	4708	390	91.72

*C*6288: 16-bit Multiplier, 32 inputs and 32 outputs			
	Total Faults	Undetected Faults	Fault Coverage (%)
LFSR	7472	101	98.65
1-d LHCA	7472	93	98.76
TLCA(I)	7472	82	98.90
TLCA(II)	7472	81	98.92
TLCA(III)	7472	92	98.77
TLCA(IV)	7472	75	99.00
TLCA(V)	7472	84	98.88

Table 5.8: Stuck-open fault simulation results for LFSR, LHCA and TLCA

to the testing time and cost. In this section, we investigate the impact of initial states of pattern generators on test coverage for the former scenario by fault simulations.

Table 5.9 lists the fault coverage of the three types of LFSMs with 10 randomly chosen initial states, where the LFSMs are length 41 machines used to generate test patterns for the 41-input ISCAS '85 benchmark circuit C1355, under the three fault models for 102,000 test patterns. Then, we apply three statistical analyses, the mean m, variance σ^2 and standard deviation σ, to the data in Table 5.9 to examine the effects. Table 5.10 summarizes these statistical results obtained by using the following equations [2]:

$$\begin{aligned} m &= \frac{1}{n_x} \sum_{i}^{n_x} x_i \\ \sigma^2 &= \frac{1}{n_x} \sum_{i} (x_i - m)^2 \\ \sigma &= \sqrt{\sigma^2} \end{aligned}$$

where n_x is the number of observed results and x_i is observed result.

The results in Table 5.10 show that (1) the initial states of the test pattern generators do not affect the test coverage of stuck-at faults, and (2) for the sequential type faults, transition and stuck-open faults, slight variations on test coverage can be found when the different initial states of the generators are used. It is understandable that stuck-at faults are least sensitive to the "randomness" of the test sequence among the three fault models. Since 102,000 test vectors are long enough to contain the patterns in the minimum test test and to detect all the faults in circuit. The sequential type faults are defined with respect to time t, therefore, they are more likely dependent on the number of transitions between two adjacent patterns and among all the patterns in a test set. The variations of test coverage using different initial states are small. In practice, the decisions on if to conduct extensive fault simulation to find the best initial state of a test generator for a given circuit should be made by test engineers based on tradeoff

Fault Model		Stuck-at	
Initial State	LFSR	LHCA	TLCA(III)
10000000000000000000000000000000000000001	99.49	99.49	99.49
11111111110000000000000000000000000000001	99.49	99.49	99.49
11111000000000000000000000000000000000001	99.49	99.49	99.49
10000000000000000000000000000001111111111	99.49	99.49	99.49
10000000000000000000000000000000000011111	99.49	99.49	99.49
11111000000000000000000000000000000011111	99.49	99.49	99.49
10000000000000011111111111000000000000001	99.49	99.49	99.49
10001110000000000000000000011111000010001	99.49	99.49	99.49
10110000000001100000000011000001100000001	99.49	99.49	99.49
10001110001110000011100011100001110000001	99.49	99.49	99.49

Fault Model		Transition	
Initial State	LFSR	LHCA	TLCA(III)
10000000000000000000000000000000000000001	96.27	98.66	98.53
11111111110000000000000000000000000000001	96.27	98.57	98.66
11111000000000000000000000000000000000001	96.27	98.57	98.66
10000000000000000000000000000001111111111	96.32	98.66	98.52
10000000000000000000000000000000000011111	96.22	98.61	98.57
11111000000000000000000000000000000011111	96.27	98.52	98.71
10000000000000011111111111000000000000001	96.22	98.47	98.61
10001110000000000000000000011111000010001	96.13	98.66	98.42
10110000000001100000000011000001100000001	96.08	98.57	98.47
10001110001110000011100011100001110000001	96.37	98.33	98.61

Fault Model		Stuck-open	
Initial State	LFSR	LHCA	TLCA(III)
10000000000000000000000000000000000000001	93.83	96.95	96.51
11111111110000000000000000000000000000001	93.64	96.82	96.88
11111000000000000000000000000000000000001	93.89	96.57	97.07
10000000000000000000000000000001111111111	93.70	96.82	96.57
10000000000000000000000000000000000011111	93.70	96.70	96.70
11111000000000000000000000000000000011111	93.70	96.63	96.76
10000000000000011111111111000000000000001	93.83	96.63	97.01
10001110000000000000000000011111000010001	93.58	96.95	96.70
10110000000001100000000011000001100000001	93.70	96.63	96.82
10001110001110000011100011100001110000001	93.58	96.32	96.70

Table 5.9: Fault coverage of three types of LFSMs with different initial states

	Stuck-at			Transition			Stuck-open		
LFSM	Mean m	Variance σ^2	Deviation σ	Mean m	Variance σ^2	Deviation σ	Mean m	Variance σ^2	Deviation σ
LFSR	99.49	0.000	0.00	96.24	0.006	0.08	93.71	0.010	0.10
LHCA	99.49	0.000	0.00	98.56	0.009	0.09	93.70	0.033	0.18
TLCA	99.49	0.000	0.00	98.57	0.007	0.08	93.77	0.017	0.16

Table 5.10: Characteristics of fault coverage distribution of three types of LFSMs

between the time of test generation and the test quality. Overall, it appears that TLCA provide the best test coverage regardless of initial states.

5.2.2.5 Summary

The five types of maximum length TLCA, TLCA(I), TLCA(II), and TLCA(III) have some desirable features in testing applications. The hardware cost of TLCA(I) is lower than that of LHCA, and only slightly higher than that of LFSRs. However, its fault coverage is inferior to that of LHCA but superior to that of LFSRs. TLCA(II) have comparable hardware costs and better fault coverage than LHCA. TLCA(III) is the most expensive type in implementation among the three, but it retains the highest average fault coverage among all types of the pseudorandom sequence generators.

In some engineering applications, hardware cost of sequence generators may not be a main concern. For example, if a LFSM is used as an external source of test vectors (rather than a built-in source), or if it is used in a communication system as a cyclic code generator, the cost of the extra XOR gates in TLCA (IV) and (V) could be negligible.

The most notable feature of primitive TLCA over all the other types is their superior performance on fault coverage at large degrees (i.e. when a machine length is ≥ 41) and at short sequence length (for example, when $T = 10{,}000$). It implies that using a TLCA as a test pattern generator can achieve high testing quality and reduce the testing time. Therefore, maximum length TLCA are a viable alternative to the conventional LFSRs and the more recent LHCA, as pseudorandom sequence generators.

In the past several years, extensive study on the pseudorandom behavior of LHCA has been reported mainly from academia. It appears that industrial practitioners have just begun to accept LHCA as an alternative or a replacement to LFSRs in testing applications. The recent study in RLCA and our study in TLCA here have increased our understanding of the structures, operations, and behavior of two-dimensional LCA. The potential applications of these machines in other fields require further investigation.

5.3 A Note on the Two-pattern Transition Property

Faults with sequential behavior such as stuck-open and delay faults need two test patterns to detect a fault. A method for assessing the two test pattern testing capabilities of linear test pattern generators is given in [17]. The paper introduced a metric called the transition coverage which indicates the number of transitions in the consecutive pairs of test patterns [39]. It was claimed the higher the transition coverage, the better the fault coverage for stuck-open and delay faults.

Definition 5.1 [39] For a given n-cell LFSM state vector $s = (s_1 s_2 \cdots s_n)$, $s_p \in \{0, 1\}$, $1 \leq p \leq n$, *a k-cell substate vector* w of s is defined by

$$w \quad = \quad (s_{i_1} s_{i_2} \cdots s_{i_k}) \tag{5.4}$$

and a *transition* corresponding to w is defined as

$$< (s_{i_1} s_{i_2} \cdots s_{i_k}), (s_{i_1}^{+} s_{i_2}^{+} \cdots s_{i_k}^{+}) >, \tag{5.5}$$

where $1 \leq i_j < i_l \leq n$ for $1 \leq j < l \leq k$.

It should be noted that even if $(s_{i_1}s_{i_2}\cdots s_{i_k}) = (s_{i_1}^+s_{i_2}^+\cdots s_{i_k}^+)$ one transition is counted because the derivation of a general equation to evaluate the number of transitions for a given substate vector is simplified.

Table 5.11 shows the test vectors generated by the LFSR(I), the LFSR(II), the LHCA and the TLCA(III) defined by the same characteristic polynomial $x^5 + x^3 + 1$. It is observed that the machines produce the same output stream in each bit position. For example, starting at $\underline{0}$ marked for each state s_i in Table 5.11, we can see that the sequences on each state s_i produced by the four LFSMs are identical.

Example 5.1 For the four LFSMs in Table 5.11, if $s = (s_1s_2s_3s_4s_5) = (00111)$ and $w = (s_2s_3s_4)$, then the transition corresponding to w is $<(s_2s_3s_4), (s_2^+s_3^+s_4^+)>$, which is

$<(011), (011)>$ for the LFSR(I) because $s^+ = (10111)$,
$<(011), (001)>$ for the LFSR(II) because $s^+ = (00011)$,
$<(011), (101)>$ for the LHCA because $s^+ = (01011)$,
$<(011), (001)>$ for the TLCA(III) because $s^+ = (10010)$.

For any particular substate vector, we can count the total number of transitions for the given substate vector for the LFSMs. For example, for $w = (s_2s_4)$ in Table 5.11, we have the following transitions:

LHCA		TLCA(III)
$<(01), (11)>$	$<(11), (01)>$	$<(10), (11)>$
$<(01), (00)>$	$<(00), (11)>$	$<(11), (00)>$
$<(11), (00)>$	$<(00), (10)>$	$<(00), (00)>$
$<(10), (00)>$	$<(11), (10)>$	$<(00), (10)>$
$<(00), (01)>$	$<(01), (10)>$	$<(11), (10)>$
$<(10), (11)>$	$<(11), (11)>$	$<(10), (01)>$
$<(00), (00)>$	$<(10), (10)>$	$<(01), (11)>$
$<(10), (01)>$	$<(01), (01)>$	$<(01), (01)>$

Time	LFSR(I)	LFSR(II)	LHCA	TLCA(III)
0	$\underline{0}0001$	$\underline{0}0001$	00001	00001
1	$1\underline{0}100$	$1\underline{0}000$	$00\underline{0}10$	01001
2	01010	$01\underline{0}00$	00111	10110
3	00101	$001\underline{0}0$	$\underline{0}1011$	00111
4	10110	$1001\underline{0}$	11001	10010
5	01011	01001	00110	$1\underline{0}111$
6	10001	10100	01001	01110
7	11100	11010	11110	00100
8	01110	01101	01101	10000
9	00111	00110	10000	$1110\underline{0}$
10	10111	10011	11000	10011
11	11111	11001	$001\underline{0}0$	11110
12	11011	11100	$0111\underline{0}$	11000
13	11001	11110	10101	00011
14	11000	11111	10100	00010
15	01100	01111	10110	$01\underline{0}11$
16	00110	00111	$1\underline{0}001$	11101
17	00011	00011	11010	11010
18	10101	10001	00011	01000
19	11110	11000	00101	11111
20	01111	01100	01100	10001
21	10011	10110	10010	10101
22	11101	11011	11111	00101
23	11010	11101	01111	$110\underline{0}1$
24	01101	01110	10111	01010
25	10010	10111	10011	10100
26	01001	01011	11101	01100
27	10000	10101	01000	01111
28	$01\underline{0}00$	01010	11100	01101
29	$001\underline{0}0$	00101	01010	$\underline{0}0110$
30	$0001\underline{0}$	00010	11011	11011
31	00001	00001	00001	00001

Table 5.11: Test vectors produced by the LFSR(I), LFSR(II), LHCA and TLCA(III) with characteristic polynomial $x^5 + x^3 + 1$

substate vector	LFSR(I)	LFSR(II)	LHCA	TLCA(III)
(s_1, s_2)	8	8	8	16
(s_1, s_3)	16	16	16	9
(s_1, s_4)	16	16	16	16
(s_1, s_5)	8	16	16	16
(s_2, s_3)	16	8	16	16
(s_2, s_4)	16	16	16	8
(s_2, s_5)	16	16	16	16
(s_3, s_4)	8	8	16	16
(s_3, s_5)	16	16	16	16
(s_4, s_5)	8	8	8	8

Table 5.12: Number of the transitions of different LFSM of degree 5

giving a total of 16 transitions for LHCA and 8 transitions for TLCA(III). Obviously, the maximum number of transitions in this case is $2^4 = 16$. The LHCA generates all the possible number of transitions, but the TLCA(III) produces only half of the maximum possible number of transitions for this substate. The complete list of all the transitions for 2-cell substate vectors for the LFSMs from Table 5.11 is given in Table 5.12. □

Definition 5.2 The *rank* of a matrix is equal to the maximum number of linear independent rows or columns.

In general, the next state w^+ of substate vector w is

$$w^+ = T_w w + T_{\bar{w}} \bar{w} \tag{5.6}$$

where $\bar{w}$ is used for notional convenience to denote a substate vector of s with cells which are not in w. A key to determining if a given k-cell substate vector w has 2^{2k} transition capability is to check whether the corresponding $T_{\bar{w}}$ has rank k [17]. It should be noted that the diagonal components of T are never included.

Example 5.2 For $w = (s_2 s_4)$ in the last example, we have

$$\begin{aligned}\begin{pmatrix} s_2^+ \\ s_4^+ \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} s_2 \\ s_4 \end{pmatrix} + \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} s_1 \\ s_3 \\ s_5 \end{pmatrix} \\ &= T_w \begin{pmatrix} s_2 \\ s_4 \end{pmatrix} + T_{\bar{w}} \begin{pmatrix} s_1 \\ s_3 \\ s_5 \end{pmatrix}\end{aligned}$$

for 1-d LHCA. The rank of $T_{\bar{w}}$ is 2, i.e. the substate is with 2^{2k} transition capability, such that the substate would be counted in Table 5.13.
But we have

$$\begin{aligned}\begin{pmatrix} s_2^+ \\ s_4^+ \end{pmatrix} &= \begin{pmatrix} 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} s_2 \\ s_4 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} s_1 \\ s_3 \\ s_5 \end{pmatrix} \\ &= T_w \begin{pmatrix} s_2 \\ s_4 \end{pmatrix} + T_{\bar{w}} \begin{pmatrix} s_1 \\ s_3 \\ s_5 \end{pmatrix}\end{aligned}$$

for TLCA(III). In this case, the rank of $T_{\bar{w}}$ is 1, i.e. the substate is without 2^{2k} transition capability such that the substate would not be counted in Table 5.13.

□

Table 5.13 lists the number of different k-cell substate vectors, which have 2^{2k} transition capability, with $k = \lfloor n/2 \rfloor$ for the LFSR, LHCA, and TLCA(III) of degrees up to 20. The data in the table for the LHCA and the LFSR are from [39]. The third column noted C_n^k gives the maximum number, which is k-combinations of n given by

$$C_n^k = \frac{n!}{k!(n-k)!} \quad .$$

The data in the table has been confirmed by direct calculation from the test pattern sequence generated.

Let $f(n)$ be the number of transitions of TLCA(III) of degree n. From the data in Table 5.13, the recurrence relations can be derived as:

$$\begin{aligned} f(n) &= f(n-1) + f(n-2) && \text{if } n = odd \\ f(n) &= 2^{\lfloor \log_2 n \rfloor} && \text{if } n = even\ . \end{aligned} \tag{5.7}$$

The number of transitions of TLCA is very close to that of LFSRs, but much less than that of LHCA. Using the transition coverage measure of the two-pattern transition property, one would consider that LHCA provides much better transition fault coverage than TLCA and LFSRs. However, our simulation results demonstrate that the transition fault coverage using TLCA as pseudorandom test pattern generator is better than using LHCA, much better than using LFSRs. This contradiction suggests that the transition property may not be a viable measure of transition fault coverage as it intended to be. Therefore, how to measure pseudorandom sequences and their test coverages remain an open and interesting problem for future research.

degree	$k = \lfloor n/2 \rfloor$	C_n^k	LFSR(I)	LFSR(II)	LHCA	TLCA(III)
3	1	3	3	2	3	3
5	2	10	7	6	8	7
7	3	35	13	10	20	11
9	4	126	21	15	48	19
11	5	462	31	21	112	27
13	6	1716	43	28	256	35
15	7	6432	57	36	576	43
17	8	24310	73	45	1280	59
19	9	92378	91	55	2816	75
2	1	2	2	2	2	2
4	2	6	3	3	4	4
6	3	20	4	4	8	4
8	4	70	5	5	16	8
10	5	252	6	6	32	8
12	6	924	7	7	64	8
14	7	3432	8	8	128	8
16	8	12870	9	9	256	16
18	9	48620	10	10	512	16
20	10	184756	11	11	1024	16

Table 5.13: Number of different k-cell substate vectors with 2^{2k} transition capability

Chapter 6

Conclusion

Linear finite state machines (LFSMs), such as linear feedback shift registers (LFSRs) and linear hybrid cellular automata (LHCA), are widely used in digital system testing, communication systems, encipherment, error-correcting coding and cryptography.

In digital testing, LFSMs are often used as pseudorandom binary sequence generators. The quality of test sequences greatly influences the ultimate quality of digital circuits and largely contributes to the overall cost of testing. This thesis addresses theoretical and practical issues of pseudorandom binary sequence generation. This research has increased our understanding of the structures, operations and pseudorandom behavior of LFSMs. The result of this research shall not be limited to testing application.

The main contributions of this thesis are:

(1) it introduces a novel implementation structure of LFSMs, named tree-structured cellular automata (TLCA). We formally define the structures and operations of TLCA, and specify thirty two linear computational rules performed by TLCA cells and the transition matrix of TLCA. We identify five types of maximum length TLCA, which are capable of generating the maximum numbers of unique binary patterns and are of great interest in engineering applications;

(2) two computer algorithms to discover maximum length TLCA are devel-

oped and the solutions to the implementation issues of the algorithms are presented. Algorithm I makes use of matrix computation and sieve methods. The complete C programming code of the implementation and its manual page are given in Appendix B.

Algorithm II employs a more direct approach, i.e. finding a maximum length machine by means of testing the primitivity of the characteristic polynomial of the transition matrix of the TLCA. It is implemented in Maple [14].

Lookup tables of low cost maximum length TLCA are provided up to degree 60. All maximum length TLCA(III) of degree 2 to 12 and their corresponding primitive characteristic polynomials are also given;

(3) a comparative study of the pseudorandom behavior of the LFSMs, linear feedback shift registers (LFSRs), linear hybrid cellular automata (LHCA), rectangular-structured linear cellular automata (RLCA) and TLCA, is conducted. The relations between the pseudorandomness of binary sequences and their effect on test coverage are investigated by computer simulation on the standard ISCAS '85 benchmark circuits.

The results of this study show that maximum length TLCA have many features desirable in engineering applications:

(1) all maximum length TLCA except the TLCA(III) preserve a comparable or lower hardware cost in implementation than that of LHCA, where TLCA(I) requires 50% less XOR gates than LHCA, and forms the most cost effective machines among one and two-dimensional linear cellular automata;

(2) TLCA of all five types provide superior or comparable test coverage to the other types of LFSMs under stuck-at, stuck-open and transition fault models for the conventional test length. In particular, TLCA demonstrate superior test coverage for stuck-open and transition faults. Moreover, the high test coverage of TLCA is best reflected at short test lengths (i.e. for test lengths $\leq 10,000$). This implies that the use of TLCA as test sequence generators can reduce test time, thus, reducing test cost while still retaining high test quality. TLCA of all types

preserve superior test coverage to conventional LFSRs under all fault models for all test length;

(3) unlike LFSRs and LHCA, the majority of TLCA do not have one-to-one correspondence with primitive polynomials, therefore, there are far fewer choices of machines at each degree. However, in engineering applications, low cost maximum length machines are often the most cost effective choice among all machines. Therefore the lack of choices may not be a problem;

(4) our simulation results suggest that the existing theoretical measures of pseudorandom sequences do not reflect the quality of the test sequences in terms of test coverage of digital circuits. For instance, the number of two-pattern transition of TLCA is comparable to that of LFSRs and much less than LHCA. Using this measurement, TLCA would be considered to have a comparable and lower test coverage than LFSRs and LHCA, respectively. Our simulation results contradict this conclusion, however, and this leads to an interesting open problem for future research.

The result of this research suggests that low cost maximum length TLCA be a viable alternative to the conventional LFSRs and LHCA as pseudorandom sequence generators. It should therefore allow digital circuits to be tested more economically and effectively.

The suggested future research includes

(1) the practical implementation of Algorithm II, and the development of more effective algorithms to find maximum length TLCA;

(2) the investigation of higher order TLCA (for example, cells with 3 or 4 children) and the other LFSM structures (for example, higher dimensional LCA, under the null and different boundary conditions);

(3) the behavior of TLCA as parallel pseudorandom array generators [5]; and

(4) the quantitative measures of test coverage and pseudorandomness of binary sequences.

Bibliography

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* Computer Science Press, 1990.

[2] M. Abramowitz. *Handbook of Mathematical Functions with Formulas, Graphs, nd Mathematical Tables*, volume I. Washington, U.S. Government Printing Office, 1964.

[3] P.H. Bardell. Analysis of Cellular Automata used as Pseudorandom Pattern Generators. In *Proceedings of the IEEE International Test Conference*, pages 762–767, 1985.

[4] P.H. Bardell. Primitive polynomials of degree 301 through 500. *Journal of Electronic Testing: Theory and Application*, 3:175–176, 1992.

[5] P.H. Bardell, W.H. McAnney, and J. Savir. *Built-In Test for VLSI: Pseudorandom Techniques.* John Wiley & Sons, 1987.

[6] C. Batut, D. Bernardi, H. Cohen, and M. Olivier. *PARI.* Version 1.39 ftped from megrez.math.u-bordeaux.fr, 1995.

[7] R.E. Blahut. *Theory and Practice of Error Control Codes.* Addison-Wesley Publishing Company, 1983.

[8] I. F. Blake, Shuhong Gao, and R. Lambert. Constructive Problems for Irreducible Polynomials over Finite Fields. In T.A. Gulliver and N.P. Secord, editor, *Information Theory.* Springer-Verlag, 1994.

[9] J. Brillhart, D. H. Lehmer, J.L. Selfridge, B. Tuckerman, and S.S. Wagstaff. *Factorizations of* $b^n \pm 1$, *b=2,3,5,6,7,10,11,12 up to high powers.* Contemporary mathematics, Vol. 22, American Mathematical Society, Providence, R. I., 1983.

[10] K. Cattell, D.M. Miller, J.C. Muzio, M. Serra, and S. Zhang. One-Dimention Linear Hydrid Cellular Automata: Synthesis, Properties, and Applications in VLSI. *Submitted to IEEE Design & Test of Computers*, 1994.

[11] K. Cattell and J.C. Muzio. Table of Linear Cellular Automata for Minimal Weight Primitive Polynomials of degree up to 300. Technical Report DCS-163-IR, Department of Computer Science, University of Victoria, Victoria, BC, Canada, June 1991.

[12] K. Cattell and J.C. Muzio. Synthesis of One-dimensional Linear Hybrid Cellular Automata. *Submitted to IEEE Transactions on Computer-Aided Design*, 1994.

[13] K. Cattell and S. Zhang. One-Dimention Linear Hydrid Cellular Automata: Synthesis, Properties, and Applications in VLSI. *Submitted to Journal of Electronic Testing: Theory and Application*, 1995.

[14] B.W. Char, K. O. Geddes, G.H. Gonnet, B.L.Leong, M.B. Monagan, and S. M. Watt. *Maple V Language Reference Manual.* Springer-Verlag, 1992.

[15] H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer-Verlag, 1993.

[16] F. Brglez and H. Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *Proc. IEEE Int. Symposium on Circuits and Systems*, pages 663–698. Special Session on ATPG and Fault Simulation, June 1985.

[17] K. Furuya and E. J. McCluskey. Two-Pattern Test Capabilities of Autonomous TPG Circuits. In *Proceedings of the IEEE International Test Conference*, pages 704–711, Oct. 1991.

[18] K.O. Geddes, S.R. Czapor, and G. Labahn. *Algorithms for Computer Algebra.* Kluwer Academic Publishers, 1992.

[19] S.W. Golomb. *Shift Register Sequences.* Aegean Park Press, Laguna Hills, CA, 1982.

[20] Torbjörn Granlund. *The GNU Multiple Precision Arithmetic Library.* Version 1.3.2 ftped from prep.ai.mit.edu, 1993.

[21] T. Hansen and G. L. Mullen. Primitive Polynomials over Finite Fields. *Mathematics of Computation*, 59:639–643, 1992.

[22] P. D. Hortensius, R. D. Mcleod, and H. C. Card. Parallel Random Number Generation for VLSI Systems Using Cellular Automata. *IEEE Transcations on Computers*, 38(10):1466–1473, Oct. 1989.

[23] H. Janoowalla. Analysis of Two-dimensional Cellular Automota. Master's thesis, Department of Computer Science, University of Victoria, Victoria, BC, Canada, 1992.

[24] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, 1969.

[25] R. Lidl and H. Niederreiter. *Finite Fields.* Encyclopedia Math. App., Vol. 20, Addison-Wesley, Reading, Mass., 1983.

[26] B. Nadeau-Dostie, D. Burek, and A. Hassan. ScanBist: A Multifrequency Scan-Based BIST Method. *IEEE Design & Test of Computers*, pages 7–17, Spring 1994.

[27] D. K. Pradhan, editor. *Fault Tolerant Computing: Theory and Techniques.* Prentice-Hall, 1986.

[28] W. Pries, A.Thanailakis, and H. C. Card. Group Properties of Cellular Automata and VLSI Application. *IEEE Transcations on Computers*, C-35(12):1013–1024, December 1986.

[29] H. Riesel. *Prime Numbers and Computer Methods for Factorization.* Birkhäuser, 1994.

[30] M. Serra, T. Slater, J.C. Muzio, and D.M. Miller. The Analysis of One-dimensional Linear Cellular Automata and their Aliasing Properties. *IEEE Transactions on Computer-Aided Design*, 9(7):767–778, July 1990.

[31] G.L. Smith. Model For Delay Faults Based Upon Paths. In *Proceedings of the IEEE International Test Conference*, pages 342–349, November 1985.

[32] Wayne Stahnke. Primitive Binary Polynomials. *Mathematics of Computation*, 27(124):977–980, October 1973.

[33] H. S. Stone. *Discrete Mathematical Structures and Their Applications.* Science Research Associates Inc., 1973.

[34] J.A. Waicukauski, E.B. Eichelberger, D.O. Forlenza, E. Lindbloom, and T. McCarthy. Fault Simulation for Structured VLSI. *VLSI Systems Design*, 6(12):20–32, December 1985.

[35] J.A. Waicukauski, E. Lindbloom, B.K. Rosen, and V.S. Iyengar. Transition Fault Simulation. *IEEE Design & Test of Computers*, 4(2):32–38, April 1987.

[36] E.J. Watson. Primitive Polynomials (mod 2). *Mathematics of Computation*, 16:368–369, 1962.

[37] S. Wolfram. Statistical Mechanics of Cellular Automata. *Reviews of Modern Physics*, 55(3):601–644, July 1983.

[38] S. Zhang, R. Byrne, J.C. Muzio, and D.M. Miller. The Evaluation of Linear Finite State Machine as Generator. Technical Report DCS-227-IR, Department of Computer Science, University of Victoria, Victoria, BC, Canada, January 1994.

[39] S. Zhang, R. Byrne, J.C. Muzio, and D.M. Miller. Why Cellular Automata are Better than LFSRs as Built-In Self-Test Generators for Sequential-type Faults. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, pages 69–72, May 1994.

[40] S. Zhang, D.M. Miller, and J.C. Muzio. Determination of Minimal Cost One-dimensional Linear Hybrid Cellular Automata. *IEE Electronics Letters*, 27(18):1625–1627, August 1991.

[41] Z. Zhang, R.D. Mcleod, D.M. Miller, and S. Zhang. Statistically Estimating Path Delay Fault Coverage in Combinational Circuits. In *Proceedings of the IEEE Pacific Rim Conference*, pages 461–464, May 1995.

Appendix A

Number of Primitive Polynomials and Irreducible Polynomials

The number of primitive polynomials of degree n is given by[25]

$$\lambda_2(n) = \frac{\phi(2^n - 1)}{n} \tag{A.1}$$

where the Euker -function is defined by

$$\phi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right) \tag{A.2}$$

where p runs through the prime dividing n. In particular, if P denotes a prime, $\phi(P) = P - 1$, if Q is also a prime, $\phi(PQ) = (P-1)(Q-1)$. Also, $\phi(P^2) = P(P-1)$.

The number of irreducible polynomials modulo 2 of degree n is given by

$$\Psi_2(n) = \frac{1}{n} \sum_{d|n} 2^d \mu\left(\frac{n}{b}\right) \tag{A.3}$$

where the sum is extended over all positive divisor d of n, and μ-function is the Mobius function. If P is a prime, $\mu(P) = -1$. If Q is also a prime, $\mu(PQ) = +1$, while $\mu(P^2) = 0$. The values of $\lambda_2(n)$ and $\Psi_2(n)$ are included in Table A.1 for $n \leq 24$.

n	$2^n - 1$	$\lambda_2(n)$	$\Psi_2(n)$
1	1	1	2
2	3	1	1
3	7	2	2
4	15	2	3
5	31	6	6
6	63	6	9
7	127	18	18
8	255	16	30
9	511	48	56
10	1,023	60	99
11	2,047	176	186
12	4,095	144	335
13	8,191	630	630
14	16,383	756	1,161
15	32,767	1,800	2,182
16	65,535	2,048	4,080
17	131,071	7,710	7,710
18	262,143	8,064	14,532
19	524,287	27,594	27,594
20	1,048,575	24,000	52,377
21	2,097,151	84,672	99,858
22	4,194,303	120,032	190,557
23	8,388,607	356,960	364,722
24	16,777,215	276,480	698,870

Table A.1: Number of primitive polynomials of degree n

Appendix B

Implementation of Algorithm I

B.1 Source Code

```
/*
 * find minimal-cost maximum-length
 * Tree-structured Linear Cellular Automata
 * using Algorithm I
 *
 * Jin Li
 * Spring, 1995
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gmp.h>


#define DEGREE 61

typedef short int matrix[DEGREE][DEGREE];

void assn(matrix m3, matrix m1, int deg);
void mat_plus(matrix m3, matrix m1, matrix m2, int deg);
void mat_mult(matrix m3, matrix m1, matrix m2, int deg);
int mat_cmp(matrix m1, matrix m2, int deg);
int irreducible(matrix m, int deg);
int primitive(matrix m, int deg);
void min1(matrix mt, int deg);
void min2(matrix mt, int deg);
void min3(matrix mt, int deg);
void min4(matrix mt, int deg);
void min5(matrix mt, int deg);
void min6(matrix mt, int deg);
void min7(matrix mt, int deg);
void min8(matrix mt, int deg);
void min9(matrix mt, int deg);
void min10(matrix mt, int deg);
void gen_mat_I(matrix m, int deg);
void gen_mat_II(matrix m, int deg);
void gen_mat_III(matrix m, int deg);
void gen_mat_IV(matrix m, int deg);
void gen_mat_V(matrix m, int deg);
void read_factor(int deg);

MP_INT factor[20];
int num_fac;            /* number of factors */
matrix mat[DEGREE+1];
FILE *ofp;

extern char *optarg;
extern int optind;

void main(int argc, char *argv[])
{
  char *ofile;
  char c;
  int deg;
  char type;

  int n, i_case;
  matrix mt;

  ofp = stdout;       /* initialization */

  /* decode arguments */
  while ((c = getopt(argc, argv, "d:t:o:n:")) != -1)
      switch (c) {
      case 'd':
        deg = atoi(optarg);
        break;
      case 't':
        type = optarg[0];
        break;
      case 'n':
        i_case = atoi(optarg);
        break;
      case 'o':
        ofile = optarg;
        ofp = fopen(ofile, "a");
        break;
      case '?':
        fprintf(stderr,
        "Usage: min -d degree -t type [-n number_one's] [-o output_file]\n");
        break;
      }

  if( argc < 5 ) {
    fprintf(stderr,
    "Usage: min -d degree -t type [-n number_one's] [-o output_file]\n");
    exit(1);
  }

  /* initialize factor[]  */
  for(n=0; n<20; n++)
    mpz_init(&factor[n]);

  read_factor(deg);           /* load  cofactors */

  /* construct proper matrix */
  if( type == '1' )
    gen_mat_I(mt, deg);
  else if( type == '2' )
    gen_mat_II(mt, deg);
  else if( type == '3' )
    gen_mat_III(mt, deg);
  else if( type == '4' )
    gen_mat_IV(mt, deg);
  else if( type == '5' )
    gen_mat_V(mt, deg);
  else {
    fprintf(stderr, "Wrong type number. 12345 only\n");
    exit(2);
  }

  /* number of 1's  */
  switch( i_case ) {
  case 1:
    min1(mt, deg); break;
  case 2:
    min2(mt, deg); break;
  case 3:
    min3(mt, deg); break;
  case 4:
    min4(mt, deg); break;
  case 5:
    min5(mt, deg); break;
  case 6:
    min6(mt, deg); break;
  case 7:
    min7(mt, deg); break;
  case 8:
    min8(mt, deg); break;
  case 9:
    min9(mt, deg); break;
  default:
    min1(mt, deg);
    min2(mt, deg);
    min3(mt, deg);
    min4(mt, deg);
    min5(mt, deg);
    min6(mt, deg);
    min7(mt, deg);
    min8(mt, deg);
    min9(mt, deg);
    min10(mt, deg);
    break;
  }

  fprintf(ofp, "No primitive found from command: ");
  for(n=0; n<argc; n++)
    fprintf(ofp, "%s ", argv[n]);
  fprintf(ofp, "\n");

  exit(0);
}


/* read_factor()
 * read in cofactors of 2^n -1
 */
void read_factor(int deg)
{
  FILE *datafile;
  int i;
  char junk[1000];

  datafile = fopen("factor.dat", "r");

  for(i=0; i<(deg-3); i++) {      /* skip (deg-3) lines */
    fgets(junk, 1000, datafile);
  }

  fscanf(datafile, "%d", &num_fac);

  for(i=0; i<num_fac; i++) {
    fscanf(datafile, "%s", junk);
    mpz_set_str(&factor[i], junk, 10);
  }

  return;
}



/* gen_mat()
 * generates proper matrix of deg for type I
 */
void gen_mat_I(matrix m, int deg)
{
  int i, j;
  int hdeg;
  int move;

  hdeg = deg / 2;
  for(i=0; i<deg; i++)
    for(j=0; j<deg; j++)
      m[i][j] = 0;

  move = 1;
  for(i=0; i<hdeg; i++) {
    move++;
    m[i][move++] = 1;
  }

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[move++][i] = 1;
    move++;
  }

  for(i=2; i<deg; i=i+2)
    m[i][i-1] = 1;

  if( deg%2 == 0 )
    m[hdeg-1][deg-1] = 1;

  return;
}

/* gen_mat()
```

```
 * generates proper matrix of deg for type II
 */
void gen_mat_II(matrix m, int deg)
{
  int i, j;
  int hdeg;
  int move;

  hdeg = deg / 2;
  for(i=0; i<deg; i++)
    for(j=0; j<deg; j++)
      m[i][j] = 0;

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[i][move++] = 1;
    m[i][move++] = 1;
  }

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[move++][i] = 1;
    move++;
  }

  for(i=2; i<deg; i=i+2)
    m[i][i-1] = 1;

  return;
}


/* gen_mat()
 * generates proper matrix of deg for type III
 */
void gen_mat_III(matrix m, int deg)
{
  int i, j;
  int hdeg;
  int move;

  hdeg = deg / 2;
  for(i=0; i<deg; i++)
    for(j=0; j<deg; j++)
      m[i][j] = 0;

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[i][move++] = 1;
    m[i][move++] = 1;
  }

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[move++][i] = 1;
    m[move++][i] = 1;
  }

  for(i=2; i<deg; i=i+2)
    m[i][i-1] = 1;

  return;
}


/* gen_mat()
 * generates proper matrix of deg for type IV
 */
void gen_mat_IV(matrix m, int deg)
{
  int i, j;
  int hdeg;
  int move;

  hdeg = deg / 2;
  for(i=0; i<deg; i++)
    for(j=0; j<deg; j++)
      m[i][j] = 0;

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[i][move++] = 1;
    move++;
  }

  move = 1;
  for(i=0; i<hdeg; i++) {
    move++;
    m[move++][i] = 1;
  }

  for(i=2; i<deg; i=i+2) {
    m[i][i-1] = 1;
    m[i-1][i] = 1;
  }

  if( deg%2 == 0 )
    m[deg-1][hdeg-1] = 1;

  return;
}

/* gen_mat()
 * generates proper matrix of deg for type V
 */
void gen_mat_V(matrix m, int deg)
{
  int i, j;
  int hdeg;
  int move;

  hdeg = deg / 2;
  for(i=0; i<deg; i++)
    for(j=0; j<deg; j++)
      m[i][j] = 0;

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[move++][i] = 1;
    m[move++][i] = 1;
  }

  move = 1;
  for(i=0; i<hdeg; i++) {
    m[i][move++] = 1;
    move++;
  }

  for(i=2; i<deg; i=i+2)
    m[i-1][i] = 1;

  return;
}


/*
 * weight 1
 */
void min1(matrix mt, int deg)
{
  int i, j;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;

    if( irreducible(mt, deg) ) {

      if( primitive(mt, deg) ) {
        for(j=0; j<deg; j++) {
          fprintf(ofp, "%d", mt[j][j]);
        }
        fprintf(ofp, " primitive degree = %d\n", deg);
        exit(0);
      }
    }

    mt[i][i] = 0;
  }

  return;
}

/*
 * weight 2
 */
void min2(matrix mt, int deg)
{
  int i, j, n;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;

      if( irreducible(mt, deg) ) {

        if( primitive(mt, deg) ) {
          for(n=0; n<deg; n++) {
            fprintf(ofp, "%d", mt[n][n]);
          }
          fprintf(ofp, " primitive degree = %d\n", deg);
          exit(0);
        }
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}


/*
 * weight 3
 */
void min3(matrix mt, int deg)
{
  int i, j, k, n;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;

        if( irreducible(mt, deg) ) {

          if( primitive(mt, deg) ) {
            for(n=0; n<deg; n++) {
              fprintf(ofp, "%d", mt[n][n]);
            }
            fprintf(ofp, " primitive degree = %d\n", deg);
            exit(0);
          }
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}


/*
 * weight 4
 */
void min4(matrix mt, int deg)
{
  int i, j, k, n;
  int s;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;
        for(s=(k+1); s<deg; s++) {
          mt[s][s] = 1;

          if( irreducible(mt, deg) ) {

            if( primitive(mt, deg) ) {
              for(n=0; n<deg; n++) {
                fprintf(ofp, "%d", mt[n][n]);
              }
              fprintf(ofp, " primitive degree = %d\n", deg);
              exit(0);
            }
          }
          mt[s][s] = 0;
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}

/*
 * weight 5
 */
void min5(matrix mt, int deg)
{
  int i, j, k, n;
  int s, t;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;
        for(s=(k+1); s<deg; s++) {
          mt[s][s] = 1;
          for(t=(s+1); t<deg; t++) {
            mt[t][t] = 1;

            if( irreducible(mt, deg) ) {

              if( primitive(mt, deg) ) {
                for(n=0; n<deg; n++) {
                  fprintf(ofp, "%d", mt[n][n]);
                }
                fprintf(ofp, " primitive degree = %d\n", deg);
                exit(0);
              }
            }
            mt[t][t] = 0;
          }
          mt[s][s] = 0;
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}


/*
 * weight 6
 */
void min6(matrix mt, int deg)
{
  int i, j, k, n;
  int s, t, p;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;
        for(s=(k+1); s<deg; s++) {
          mt[s][s] = 1;
          for(t=(s+1); t<deg; t++) {
            mt[t][t] = 1;
            for(p=(t+1); p<deg; t++) {
              mt[p][p] = 1;

              if( irreducible(mt, deg) ) {
                if( primitive(mt, deg) ) {
                  for(n=0; n<deg; n++) {
                    fprintf(ofp, "%d", mt[n][n]);
                  }
                  fprintf(ofp, " primitive degree = %d\n", deg);
                  exit(0);
                }
              }
              mt[p][p] = 0;
            }
            mt[t][t] = 0;
          }
          mt[s][s] = 0;
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}



/*
 * weight 7
 */
void min7(matrix mt, int deg)
{
  int i, j, k, n;
  int s, t, p, q;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;
        for(s=(k+1); s<deg; s++) {
          mt[s][s] = 1;
          for(t=(s+1); t<deg; t++) {
            mt[t][t] = 1;
            for(p=(t+1); p<deg; t++) {
              mt[p][p] = 1;
              for(q=(p+1); q<deg; q++) {
                mt[q][q] = 1;

                if( irreducible(mt, deg) ) {

                  if( primitive(mt, deg) ) {
                    for(n=0; n<deg; n++) {
                      fprintf(ofp, "%d", mt[n][n]);
                    }
                    fprintf(ofp, " primitive degree = %d\n", deg);
                    exit(0);
                  }
                }
                mt[q][q] = 0;
              }
              mt[p][p] = 0;
            }
            mt[t][t] = 0;
          }
          mt[s][s] = 0;
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}

/*
 * weight 8
 */
```

```
void min8(matrix mt, int deg)
{
  int i, j, k, n;
  int s, t, p, q, a;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;
        for(s=(k+1); s<deg; s++) {
          mt[s][s] = 1;
          for(t=(s+1); t<deg; t++) {
            mt[t][t] = 1;
            for(p=(t+1); p<deg; t++) {
              mt[p][p] = 1;
              for(q=(p+1); q<deg; q++) {
                mt[q][q] = 1;
                for(a=(q+1); a<deg; a++) {
                  mt[a][a] = 1;

                  if( irreducible(mt, deg) ) {
                    if( primitive(mt, deg) ) {
                      for(n=0; n<deg; n++) {
                        fprintf(ofp, "%d", mt[n][n]);
                      }
                      fprintf(ofp, " primitive degree = %d\n", deg);
                      exit(0);
                    }
                  }
                  mt[a][a] = 0;
                }
                mt[q][q] = 0;
              }
              mt[p][p] = 0;
            }
            mt[t][t] = 0;
          }
          mt[s][s] = 0;
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}


/*
 * weight 9
 */
void min9(matrix mt, int deg)
{
  int i, j, k, n;
  int s, t, p, q, a, b;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;
        for(s=(k+1); s<deg; s++) {
          mt[s][s] = 1;
          for(t=(s+1); t<deg; t++) {
            mt[t][t] = 1;
            for(p=(t+1); p<deg; t++) {
              mt[p][p] = 1;
              for(q=(p+1); q<deg; q++) {
                mt[q][q] = 1;
                for(a=(q+1); a<deg; a++) {
                  mt[a][a] = 1;
                  for(b=(a+1); b<deg; b++) {
                    mt[b][b] = 1;

                    if( irreducible(mt, deg) ) {
                      if( primitive(mt, deg) ) {
                        for(n=0; n<deg; n++) {
                          fprintf(ofp, "%d", mt[n][n]);
                        }
                        fprintf(ofp, " primitive degree = %d\n", deg);
                        exit(0);
                      }
                    }
                    mt[b][b] = 0;
                  }
                  mt[a][a] = 0;
                }
                mt[q][q] = 0;
              }
              mt[p][p] = 0;
            }
            mt[t][t] = 0;
          }
          mt[s][s] = 0;
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}


/*
 * weight 10
 */
void min10(matrix mt, int deg)
{
  int i, j, k, n;
  int s, t, p, q, a, b, c;

  for(i=0; i<deg; i++) {
    mt[i][i] = 1;
    for(j=(i+1); j<deg; j++) {
      mt[j][j] = 1;
      for(k=(j+1); k<deg; k++) {
        mt[k][k] = 1;
        for(s=(k+1); s<deg; s++) {
          mt[s][s] = 1;
          for(t=(s+1); t<deg; t++) {
            mt[t][t] = 1;
            for(p=(t+1); p<deg; t++) {
              mt[p][p] = 1;
              for(q=(p+1); q<deg; q++) {
                mt[q][q] = 1;
                for(a=(q+1); a<deg; a++) {
                  mt[a][a] = 1;
                  for(b=(a+1); b<deg; b++) {
                    mt[b][b] = 1;
                    for(c=(b+1); c<deg; c++) {
                      mt[c][c] = 1;

                      if( irreducible(mt, deg) ) {
                        if( primitive(mt, deg) ) {
                          for(n=0; n<deg; n++) {
                            fprintf(ofp, "%d", mt[n][n]);
                          }
                          fprintf(ofp, " primitive degree = %d\n", deg);
                          exit(0);
                        }
                      }
                      mt[c][c] = 0;
                    }
                    mt[b][b] = 0;
                  }
                  mt[a][a] = 0;
                }
                mt[q][q] = 0;
              }
              mt[p][p] = 0;
            }
            mt[t][t] = 0;
          }
          mt[s][s] = 0;
        }
        mt[k][k] = 0;
      }
      mt[j][j] = 0;
    }

    mt[i][i] = 0;
  }

  return;
}


/* assn()
 * assign one matrix to another
 */
void assn(matrix m3, matrix m1, int deg)
{
  int i, j;

  for(i=0; i<deg; i++) {
    for(j=0; j<deg; j++) {
      m3[i][j] = m1[i][j];
    }
  }

}

/* mat_plus()
 * matrix addition
 */
void mat_plus(matrix m3, matrix m1, matrix m2, int deg)
{
  int i, j;

  for(i=0; i<deg; i++) {
    for(j=0; j<deg; j++) {
        m3[i][j] = m1[i][j] ^ m2[i][j];
    }
  }

}

/* mat_mult()
 * matrix multiplication
 */
void mat_mult(matrix m3, matrix m1, matrix m2, int deg)
{
  int i, j, n;

  for(i=0; i<deg; i++) {
    for(j=0; j<deg; j++) {
      m3[i][j] = 0;

      for(n=0; n<deg; n++) {
        if( m1[i][n] )
          m3[i][j] ^= m2[n][j];
      }
    }
  }

}


/* mat_cmp()
 * compares the equality of two matrices
 * return 0   not equal
 * return 1   equal
 */
int mat_cmp(matrix m1, matrix m2, int deg)
{
  int i, j;

  for(i=0; i<deg; i++) {
    for(j=0; j<deg; j++) {
      if( m1[i][j] != m2[i][j] )
        return 0;
    }
  }

  return 1;
}


/* irreducible()
 * check whether input matrix is irreducible
 * return 1:      irreducible
 * return 0:       reducible
 */
int irreducible(matrix m, int deg)
{
  int i;

  assn(mat[0], m, deg);

  /*
   * i=1,..., deg-1, see M^{2^{i}} = M  -> reducible
   */
  for(i=1; i<deg; i++) {
    mat_mult(mat[i], mat[i-1], mat[i-1], deg);

    if( mat_cmp(mat[i], m, deg) ) {
      return 0;                              /* reducible matrix */
    }
  }

  /* i=deg */
  mat_mult(mat[deg], mat[deg-1], mat[deg-1], deg);

  if( mat_cmp(mat[deg], m, deg) )
    return 1;
  else
    return 0;

}

/* primitive()
 * test primitive after testing irreducible
 */
int primitive(matrix m, int deg)
{
  int i, j;
  int len;
  int n_power;
  int p[20];
  char str[1000];

  matrix m_current, m_t;

  for(i=0; i<num_fac; i++) {
    n_power = 0;
    mpz_get_str(str, 2, &factor[i]);
    len = strlen(str);

    for(j=0; j<len; j++) {
      if( str[j] == '1' )
        p[n_power++] = len - j - 1;
    }

    assn(m_current, mat[ p[0] ], deg);
    for(j=1; j<n_power; j++) {
      mat_mult(m_t, m_current, mat[ p[j] ], deg);
      assn(m_current, m_t, deg);
    }
    if( mat_cmp(m_current, m, deg) ) {
      return 0;
    }
  }

  return 1;
}
```

B.2 Manual Page

NAME

min – the program to search minimal-cost maximum length TLCA

SYNOPSIS

min -d degree -t type [-o outfile]

DESCRIPTION

The software is an implementation of Algorithm I. The program finds minimal-cost maximum length tree-structured linear cellular automata.

OPTIONS

-d *degree*	input degree of TLCA requested.
-t *type*	choose certain type of TLCA requested, where type is a number from 1 to 5, corresponding to certain types of TLCA. 1 – Type I TLCA. 2 – Type II TLCA. 3 – Type III TLCA. 4 – Type IV TLCA. 5 – Type V TLCA.
-o *outfile*	send the output produced to the file *outfile.* By default, the output is written to the standard output *stdout.*

EXAMPLE

```
min -d 41 -t 3
```

gives the type III of minimal-cost maximum length TLCA of degree 41.

Appendix C

User Manual for the Fault Simulator

C.1 Manual Page

"User's Manual of Parallel Pattern Single Fault Propagation simulator"
developed by Mr. S. Zhang
Department of Computer Science, University of Victoria

NAME

sim3 – a combinational logic fault simulator

SYNOPSIS

sim3 [options] circuit_file

DESCRIPTION

Read in a description of a combinational circuit from a **circuit_file** with the ISCAS [16] netlist format produced by OASIS (an Open Architecture Silicon Implementation Software). Simulate fault-free, single stuck-at faults, and/or single delay faults and/or single transistor stuck-open faults for a given pseudorandom test pattern generator implemented by either a CA, LFSR, XCA or Binary-Counter. The test patterns can also be from a file or the standard input. **sim3** evaluates exact fault coverage and the number of undetected faults. The fault coverage is given by ((N-U)/N)*100 where N is the number of total faults of the type being considered and U is the number of undetected faults.

The implementation of **sim3** is based on the PPSFP (parallel Pattern Single Fault Propagation) technique, and the comprehensive fault equivalence rules for fault collapsing are applied. Before simulating stuck-open faults, **sim3** replaces XOR, AND and OR gates with NAND, NOR and NOT gates in the circuit under test. The CA considered is one-dimensional linear cellular automata. Two kinds of CA are used, namely CA with minimum cost and CA corresponding to the LFSR with minimum cost. The types of linear feedback shift register (LFSR) used are the LFSR with minimum cost, LFSR with half taps evenly distributed and LFSR similar to the CA with minimum cost.

sim3 can also produce the circuit netlist with the local format and a truth table with the *espresso* format.

OPTIONS

-V	Produce circuit file with local format.
-F	Produce a truth table with the *espresso* format.
-R	Replace XOR, AND and OR with NOT, NAND and NOR gates in the circuit under test.
-S	Simulate Stuck-at faults.
-D	Simulate Delay faults.
-O	Simulate Stuck-Open faults (imply option **-R**).
-Gn	Choose test pattern generator, where n is a number from the range 0 to 5, corresponding to the following types of the test pattern generator: 0 – CA with minimum cost (default).

1 – CA corresponding to the LFSR with minimum cost.
2 – LFSR with minimum cost.
3 – LFSR with half taps evenly distributed.
4 – LFSR corresponding to the CA with minimum cost.
5 – Binary Counter.

If **-Gn** is not specified, then the default test pattern generator is taken to be the CA with minimum cost.

-I	Used for specifying Type I LFSR (XORs are between the cells). By default, Type II LFSR (XORs are on the feedback chain) will be used. This option only works with the option **-G2**.
-X	Used for specifying the XCA. This option only works with the option **-G0**.
-Q	Include all zeros pattern for the CA and LFSR generator.
-i *seed*	Set initial random seed for the generator. The *seed* is a positive number for producing a random value.
-l *length*	Set test sequence length. By default, the length of sequence is 2^m, where m is the number of inputs in the circuit. If $m > 30$, this option must be specified.
-v *vectorfile*	Test patterns are from the *vectorfile*. If using option **-v** +, the test patterns are read from the standard input *stdin*. If the **-v** option is specified, then the option **-Gn** will be ignored.
-T	Produce fault coverage for each 256 test patterns.
-o *outfile*	send the output produced to the file *outfile*. By default, the output is written to the standard output *stdout*.

Appendix D

Data of Test Length vs Fault Coverage

This appendix lists the data of the stuck-at fault, delay fault and stuck-open fault coverage as a function of number of test patterns. The test pattern generators include LFSR(II), LHCA, TLCA(I), TLCA(II) and TLCA(III). The plots of the data are illustrated in Figures 5.6, 5.7, 5.8, 5.9, 5.10, and 5.11 of Chapter 5.

*C*1355: ECAT 41 inputs and 32 outputs					
Fault Model			STUCK-AT		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	77.32	76.24	88.25	89.39	87.99
500	89.64	92.38	95.43	96.12	94.79
700	94.60	93.20	96.76	97.78	96.32
1000	98.03	95.81	98.28	99.30	97.65
2000	99.36	99.30	99.49	99.40	99.49
4000	99.49	99.49	99.49	99.49	99.49
5000	99.49	99.49	99.49	99.49	99.49
7000	99.49	99.49	99.49	99.49	99.49
102000	99.49	99.49	99.49	99.49	99.49
Fault Model			DELAY		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	47.75	38.91	71.51	75.67	70.46
500	69.89	82.31	88.91	90.11	86.57
700	79.59	84.61	91.20	91.68	90.15
1000	86.04	89.29	93.83	93.88	92.78
2000	92.02	95.65	96.99	97.04	96.70
4000	94.93	97.51	97.85	98.04	97.94
5000	95.60	97.85	97.99	98.04	98.09
7000	95.79	97.90	98.14	98.04	98.09
10000	95.79	98.04	98.14	98.04	98.09
20000	95.94	98.04	98.28	98.04	98.14
60000	96.13	98.28	98.47	98.23	98.28
102000	96.22	98.33	98.56	98.42	98.37
Fault Model			STUCK-OPEN		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	45.70	34.79	64.28	68.20	63.34
500	63.47	76.62	82.17	83.98	81.67
700	71.32	79.11	84.66	85.35	84.23
1000	77.06	83.73	89.52	87.59	86.47
2000	84.41	89.21	91.02	91.77	91.15
4000	89.84	91.96	93.14	93.33	92.77
5000	91.08	92.39	93.39	93.45	93.08
10000	92.02	93.64	94.14	94.39	93.52
20000	92.58	94.70	95.07	95.14	94.58
60000	93.27	95.82	96.38	96.26	96.07
102000	93.39	96.26	96.38	96.51	96.51

Table D.1: Fault coverages at different number of test patterns

*C*3540: ALU and Control 50 inputs and 22 outputs					
Fault Model			STUCK-AT		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	51.40	76.72	81.53	82.44	79.64
500	90.34	90.43	91.45	92.01	91.48
1000	94.71	94.11	94.22	94.75	94.19
2000	95.60	95.54	95.33	95.60	95.48
4000	95.71	95.83	95.71	95.80	95.83
5000	95.80	95.89	95.83	95.85	95.92
7000	95.89	95.95	95.89	95.86	95.92
10000	95.92	95.97	95.89	95.89	95.95
20000	95.97	95.97	95.97	95.92	95.97
60000	96.00	96.00	96.00	96.00	96.00
102000	96.00	96.00	96.00	96.00	96.00
Fault Model			DELAY		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	28.79	60.40	68.64	67.71	62.91
500	74.48	79.80	83.60	84.85	91.48
1000	82.96	86.30	88.78	89.78	88.02
2000	87.51	90.80	92.18	92.84	91.86
4000	88.97	92.54	94.07	94.29	92.95
5000	89.25	92.97	94.50	94.67	93.07
7000	89.48	93.16	94.73	94.86	93.20
10000	89.57	93.29	94.77	95.07	93.27
20000	89.67	93.33	94.99	95.16	93.29
60000	89.72	93.48	95.05	95.24	93.44
102000	89.72	93.48	95.05	95.28	93.46
Fault Model			STUCK-OPEN		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
100	24.58	51.02	58.14	57.03	53.02
500	65.78	71.05	75.15	77.17	73.26
1000	74.51	79.23	81.71	82.75	80.37
2000	79.80	84.71	86.53	87.00	86.41
4000	81.86	87.23	89.36	89.51	88.83
5000	82.18	87.96	90.00	90.14	89.21
7000	82.52	88.62	90.70	90.74	89.95
10000	82.73	89.08	91.08	91.25	90.42
20000	83.01	89.51	91.93	92.01	90.74
60000	83.18	89.93	92.95	92.80	91.23
102000	83.26	90.02	93.12	92.97	91.38

Table D.2: Fault coverages at different number of test patterns

$C$880: ALU and Control 60 inputs and 26 outputs					
Fault Model			STUCK-AT		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
200	45.75	91.19	94.90	94.06	93.52
400	53.29	95.44	96.60	96.50	95.97
500	58.39	96.60	96.71	96.82	96.39
700	63.91	97.24	97.24	96.82	96.92
1000	77.60	97.77	97.98	97.35	97.45
2000	88.96	98.62	99.26	98.94	99.04
5000	96.18	99.04	99.56	99.58	99.58
10000	99.47	99.58	99.89	99.58	99.89
20000	99.79	99.88	99.90	99.79	99.89
60000	100.0	100.0	100.0	100.0	100.0
102000	100.0	100.0	100.0	100.0	100.0
Fault Model			DELAY		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
200	19.27	82.94	88.65	87.81	86.14
400	26.50	89.87	92.00	92.00	90.78
500	30.92	91.32	92.46	92.16	86.78
700	37.17	92.16	93.68	92.61	92.99
1000	49.43	93.75	95.66	93.37	93.91
2000	67.33	95.96	97.41	96.19	97.03
5000	86.06	96.57	98.48	98.10	98.17
10000	95.43	98.25	99.31	98.86	98.63
20000	95.89	99.24	99.54	99.16	98.93
60000	96.34	99.47	99.77	99.62	99.31
102000	96.34	99.47	99.77	99.62	99.31
Fault Model			STUCK-OPEN		
Length	LFSR	LHCA	TLCA(III)	TLCA(II)	TLCA(I)
200	17.89	78.65	82.28	81.24	81.16
400	25.50	85.31	86.00	86.78	86.17
500	30.92	87.12	86.95	87.64	86.78
700	37.86	88.25	88.42	92.61	88.42
1000	48.40	89.71	90.06	89.71	89.46
2000	66.12	93.00	93.43	92.83	93.26
5000	83.75	93.78	94.90	94.99	94.38
10000	87.99	95.25	95.28	96.02	95.25
20000	93.78	96.02	96.80	97.23	96.37
60000	94.64	96.54	97.32	97.93	97.67
102000	94.64	96.63	97.58	98.01	97.84

Table D.3: Fault coverages at different number of test patterns

Appendix E

The LFSRs, LHCA and TLCA in Fault Coverage Simulations

This appendix lists the actual LFSMs, LFSR(II), LHCA, and types I to V TLCA, used in the fault coverage simulations of Chapter 5. Their binary string representations are given.

LFSM	Degree	Representation of binary string
LFSR	32	100011000000000000000000000000011
LFSR	33	1000000000000000000010000000000001
LFSR	36	1000000000000000000000000100000000001
LFSR	41	100000000000000000000000000000000000001001
LFSR	50	100000000000000000000001100000000000000000000000011
LFSR	60	1000000000000000000000000000000000000000000000000000000000011
LHCA	32	10000000000000000000000010000000
LHCA	33	100000000000000000000000000000000
LHCA	36	000001000000000000000000000000000000
LHCA	41	10000000000000000000000000000000000000000
LHCA	50	00000000001000000000000000000000000000000000000000
LHCA	60	010000000000000000000000000000000000010000000000000000000000
TLCA(I)	32	10000001000000010000000000000001
TLCA(I)	33	100001000000000000000000000000011
TLCA(I)	36	110000010000000001000000000000000001
TLCA(I)	41	10000001100000000000000000000000000000011
TLCA(I)	50	00100010000000000000000000110000000000000000000000
TLCA(I)	60	000000010000000000000000000100000000000000000000000000000001
TLCA(II)	32	10100001100000000000000000000001
TLCA(II)	33	101000010000000110000000000000000
TLCA(II)	36	000110000000000000000000000000000001
TLCA(II)	41	00100000000000000001100000000000000000000
TLCA(II)	50	10000010000000000000000000110000000000000000000000
TLCA(II)	60	100000000000000000000000000000100000000000000000000000000000
TLCA(III)	32	10010001000000001000000000000000
TLCA(III)	33	100100000000000010000000000000000
TLCA(III)	36	100100010000000000100000000000000000
TLCA(III)	41	10010000010000000000100000000000000000000
TLCA(III)	50	10000000000110000000000000000000000000000000000001
TLCA(III)	60	000000000000000000000000000001000000000000000000000000000000
TLCA(IV)	32	00100000100000000000000000000001
TLCA(IV)	33	000000001000000000000000000000000
TLCA(IV)	36	001101000000000000000000000000000001
TLCA(IV)	41	00000000001000000000000000000000000000000
TLCA(IV)	50	00100000000010000000000010000000000000000000000000
TLCA(IV)	60	100101000000000100000000000000000000000000000000000000000001
TLCA(V)	32	10100001100000000000000000000001
TLCA(V)	33	101000010000000110000000000000000
TLCA(V)	36	000110000000000000000000000000000001
TLCA(V)	41	00100000000000000001100000000000000000000

Table E.1: The LFSMs used in fault coverage simulations in Chapter 5